T.P.E.D. PR9

Projet ORCHID :

Plate-forme de Télétravail
à
Système Réparti Multi-Agents Intelligents

Suivi assuré par Monsieur Philippe BENHAMOU (ONERA)

DARCHE Marc-Aurèle PIERRE Emmanuel SALVY Frédéric
SOULIEZ Antoine VARROY Marc-Olivier

10 mars 1999

Table des matières

Table des figures

Introduction

Les entreprises tendent au cours de leur développement à s'internationnaliser, à développer des projets sur plusieurs sites, dans plusieurs pays, et ainsi en va-t-il pour toute tâche nécessitant un travail collaboratif.
Cela va d'une échelle locale, un groupe de développeurs, à un ensemble plus global, une entreprise de dimension internationale.
Tout développement soulève la problématique du déploiement d'un *Système d'Information*.
Quand une multinationale se lance dans le déploiement d'un applicatif, cela se chiffre en millions de francs, aussi bien en logiciels qu'en installation et maintenance.
Un autre coût majeur est le temps passé pour la transmission et la recherche d'informations.

Un des intérêts des nouvelles technologies est de simplifier ces tâches au travers de programmes conviviaux en mobilisant le moins possible les utilisateurs.
Les applications répondant à ces besoins se doivent d'être portables.
Pour cela elles doivent utiliser des protocoles standards (les standards Internet par exemple), et non des standards propriétaires comme c'est le cas pour beaucoup de solutions actuelles (comme pour les produits Microsoft par exemple).
Un autre point à prendre en compte est le facteur humain. Comment les nouveaux outils modifient-ils l'organisation du travail, le *WorkFlow* ?
Enfin, quelle appréhension les utilisateurs ont-ils vis à vis de ces nouveaux outils ?
Un des buts de notre projet de TPED, est de tenir compte de ces facteurs.

Concernant nos recherches, 98% des livres [23, 25, 39, 40, 47, 51, 52], documentations [16–21, 27, 30, 32, 35, 36, 41], sites internets [1, 2, 4, 5, 7, 9–15] et 100% des programmes que nous utilisons pour ce projet sont en langue anglaise.
Cela n'a rien d'étonnant car il est facile de constater que l'Anglais est la langue de l'Informatique et des Technologies de l'Information (IT).
De ce fait, nous avons en quelque sorte "pensé" notre projet en anglais. Nous avons donc pris le parti de garder la langue anglaise pour tout ce qui est d'ordre technique dans ce rapport. Ainsi, la présence de termes techniques, d'acronymes et de schémas en anglais devient tout à fait naturelle.
On notera aussi la présence d'un glossaire majoritairement écrit en anglais.
Enfin, en ce qui concerne les interfaces de nos programmes, elles présentent toutes des dialogues et informations en anglais. C'est en effet le meilleur parti à prendre si l'on souhaite que notre travail puisse être utilisé de manière internationale.

Chapitre 1

Présentation du projet

Ce projet se déroule dans le cadre des T.P.E.D. (Travaux Pratiques d'Etudes et de Développement) de 3e année de cycle ingénieur à l'E.S.I-E-A.

Ce projet bénéficie d'une enveloppe horaire d'un peu plus de 2 mois. Les périodes de travail sur le projet ne sont pas consécutives. Mais 2 mois ne représentent qu'une trop petite période pour des projets de grande envergure.

Le suivi de ce projet est assuré par Philippe BENHAMOU (benhamou@onera.fr), chercheur à l'Office National d'Études et de Recherches Aérospatiales (ONERA).
L'ONERA est un établissement public de recherche, scientifique et technique, à caractère industriel et commercial. L'ONERA est sous tutelle du ministère de la Défense.
L'ONERA a pour vocation de développer et d'orienter les recherches dans le domaine aérospatial, de concevoir, réaliser et mettre en oeuvre les moyens nécessaires aux recherches et à la conduite des essais, de diffuser et valoriser les résultats des recherches ainsi que de concourir à la formation des ingénieurs.
Son site web est http://www.onera.fr

Notre **Plateforme de Télétravail à Système Réparti Multi-Agents Intelligents** est un ensemble Client/Serveur portable et évolutif présentant les spécificités suivantes :

- Le contenu informatif et applicatif est réparti sur plusieurs serveurs.
- Chaque serveur dispose d'agents intelligents et autonomes pour recueillir les informations et assurer la liaison avec ses clients.
- Les serveurs communiquent entre eux et avec les clients par le protocole http, ce qui ouvre les applicatifs à tout le Web, même derrière des bastions (Firewalls).

Les clients peuvent échanger et partager documents et Bases de Données (BdD). Ils peuvent travailler simultanément.
Les clients peuvent étendre et mettre à jour l'ensemble de leurs outils et agents.
Les agents peuvent être programmés pour exécuter des tâches spécifiques.
Les serveurs sont capables d'échanger entre eux leurs agents ainsi que ceux de leurs clients.

Les connaissances stockées dans des Bases de Données peuvent être échangées via le langage KQML.
Tous les programmes seront développés en Java et/ou Perl.

Java permet à chacun des membres de l'équipe de développer séparément puis d'intégrer son travail à l'ensemble, comme étudié en Génie Logiciel et Programmation Orientée Objet (GLPOO).

Pour ce projet, on se propose de réaliser les agents suivants:

- Agent personnel

- Agent prise de Rendez-vous (RDV)
- Agent recherche d'informations sous forme d'URLs (cf SoftBot [29])
- Agent d'échange d'informations
- Agent ardoise à dessin
- Agent de recherche de personnes connectées
- Agent Post-it
- Agent maintenance

Nous développons en priorité l'Agent prise de Rendez-vous

Chapitre 2

Le facteur humain

Un facteur souvent négligé

Lors de la création d'un produit le facteur humain est souvent négligé. Il apparaît que dans l'avenir, les systèmes d'information sont appelés à devenir une technologie complètement intégrée, naturelle pour tous. L'Homme et la Machine vont donc sans cesse interagir.

L'ergonomie d'un produit est souvent déterminante pour sa réussite. L'Interface Homme Machine (IHM) se situe à la confluence des sciences et des techniques informatiques et psychologiques, considérant qu'il est essentiel pour réaliser une interface (assemblage de composants matériels et logiciels) de prendre en considération l'ensemble des phénomènes physiques et cognitifs qui interviennent dans la communication entre l'homme et la machine.
Mais l'ergonomie n'est pas tout, il faut aussi prendre en compte les besoins et surtout les habitudes et façons de penser des utilisateurs. Ce n'est pas évident car chaque personne est unique, mais il faut faire un effort de standardisation.

Un produit doit être convivial s'il veut s'adresser au plus grand nombre.

En effet les utilisateurs ont des habitudes et travaillent selon des protocoles (protocoles de travail). Même si le produit demande de travailler d'une manière différente, il faut tenir compte des habitudes passées des utilisateurs de manière à leur éviter à passer du temps dans la manipulation du nouveau produit, afin de pas réduire leur productivité.

Lorsqu'on conçoit un nouveau produit, il faut savoir que pour les utilisateurs il y aura :

- Modification des protocoles de travail.
- Modification des habitudes.
- Modification de la productivité.

Dans le secteur informatique, la principale erreur des concepteurs est qu'ils oublient que les utilisateurs ne sont pas, la plupart du temps, des informaticiens. Les concepteurs oublient souvent de se mettre à la place des utilisateurs. Lorsqu'on conçoit un produit, il faut 'essayer' de partir avec un esprit vierge, de même lors des tests de convivialité du produit.

Cela nécessite une culture générale informatique et une connaissance de l'organisation des entreprises. En effet, il faut synthétiser les domaines suivants :

- les études des besoins
- l'écoute des utilisateurs et le cahier des charges
- fonctionnalités nouvelles attendues
- volume, utilisation, circulation par mois et par an

- les choix techniques, matériels et logiciels qui en découlent
- la reprise de l'existant
- les formations
- mesure et établissement d'un plan qualité

Prise en compte du facteur humain

Nous nous attachons surtout à la productivité.

Notre interface sera très ergonomique et la conception de notre produit est conçue de manière à améliorer la productivité des utilisateurs en permettant une consultation et un échange plus facile et plus rapide de l'Information.

Ainsi, par exemple, la place de la souris sera matérialisée sur le bureau virtuel qu'aura chaque client. Nous utilisons le module graphique Swing de Java qui permet d'inclure des objets graphiques standards et intuitifs : boutons, barres défilantes, onglets, etc...

Méthodologie SAAS

La méthode SAAS([24])a été développée pour déterminer les services qu'il faut offrir à une population de différents utilisateurs. Elle a été élaborée en gardant à l'esprit deux ensembles de contraintes: celles techniques, et celles humaines.

Contraintes techniques

- les services cruciaux doivent être reconnus par tous les utilisateurs.
- le savoir doit être propagé parmi tous les agents avec une granularité optimale. D'ailleurs, si le savoir est trop complexe, alors les agents seront complexes, difficiles à implémenter et à controler, d'autre part, si le savoir est trop ténu, alors les communications seront complexes et coûteuses.
- puisque les agents fournissent des services, il est important que les dépendances entre les services soient visibles, ce qui peut aider en cas d'echec.

Contraintes humaines

- les interviews doivent être courtes.
- les points de vue doivent être facilement représentés pour une meilleure identification des activités critiques.
- la représentation doit supporter différents niveaux de détails.
- le formalisme utilisé doit être simple et intuitif.
- le formalisme ne doit pas inclure de biais.

La méthode SAAS

Pour donner un apperçu, voici les étapes:

- récolter les points de vue.
- classifier les activités et les ressources.
- validation par le groupe.
- description des services.
- écriture des scénarios.
- construire une maquette.
- identifier les compétences.

– synthétiser les compétences.

Les détails et résultats sont disponibles dans [24] pp12,13.

Il faut accompagner et favoriser le changement de comportement des utilisateurs et savoir anticiper les différentes techniques organisationnelles et culturelles pour assurer le succès de l'implémentation de ces types d'organisations au sein de l'entreprise.

En conclusion, il aurait été bon d'avoir un psychologue et/ou un ergonome pour la conception de la partie destinée au client.

Chapitre 3

L'IAD dans le projet

3.1 Présentation

L'*Intelligence Artificielle Distribuée* (IAD), ou encore DAI en Anglais, naît aux environs de 1970. Suivent ensuite beaucoup de travaux dans les années 1980. Les premières applications informatiques voient le jour en 1990.

Il ne faut pas confondre l'*Intelligence Artificielle* (IA), ou AI en Anglais, avec l'IAD.

En effet l'IA est en 1998, déjà utilisée depuis longtemps dans beaucoup de produits finis, même commerciaux. On trouve de l'IA dans presque tous les simulateurs actuels par exemple.

Ce n'est pas le cas de l'IAD, qui est un domaine relativement récent.
Cette spécialité attire les chercheurs et déchaîne la créativité. Mais comme il nous a été donné de nous en apercevoir au cours de notre travail, ce domaine en est à ses débuts, dans le sens où tous les projets existants, et ils sont légions, sont tous à l'état de recherche. Aucun n'a jusqu'à présent fait l'objet d'une commercialisation.

Il nous a paru très intéressant d'inclure de l'IAD dans notre projet pour le rendre plus riche et à la fois pour nous préparer à ce qui sera très certainement au coeur de la plupart des systèmes d'informations de demain.
L'intégration de l'IAD dans le projet a été l'une des parties les plus intéressantes du projet.

En IAD les tâches sont accomplies par des agents.
Pour une analogie rapide on peut penser aux fourmis, dont le travail en tant qu'individu est insignifiant, mais dont la fourmillière, auto-organisatrice, est toute puissante. Les fourmis sont les agents du système fourmillière!

Mais il ne s'agit vraiment ici que d'une très rapide analogie, l'IAD étant en effet divisée en 2 domaines principaux:

1. Les Systèmes Multi-Agents (SMA)
2. La Résolution de Problèmes Distribués (RPD)

De plus le mot "Agent" est compris quelque peu différemment selon les chercheurs et développeurs auxquels on s'adresse. Voilà pourquoi il est essentiel de discuter des différentes acceptations du terme "agent".

3.2 Les différentes acceptations du terme "agent"

Nous allons voir qu'il y a 2 grandes acceptations et compréhensions du mot agent.

La communauté IAD
La communauté de l'Intelligence Artificielle Distribuée fut la toute première à utiliser le terme agent.

L'agent, du point de vue IAD, doit satisfaire aux conditions suivantes:

- C'est une entité, que ce soit logicielle, animale, ou humaine
- Il est autonome
- Il agit sur lui-même et/ou son environement
- Il est capable de prendre des initiatives et d'agir sans qu'on lui ait dit de le faire
- Il n'a de sens que parce qu'il interagit avec d'autres agents. Quand on parle d'agent, c'est en fait "d'agents" dont on parle. Il est sous entendu qu'il y en a plusieurs, i.e. que l'intelligence n'est plus la propriété d'un agent mais du système d'agents.

Au sein de la communauté IAD, il est admis qu'il y a 2 types d'agents: Les agents *réactifs*, et les agents *cognitifs*.

1. Les agents réactifs:
   - Ils réagissent à des stimuli.
   - Leur langage est limité.
   - On peut les comparer aux fourmis.
2. Les agents cognitifs:
   - Ils effectuent un traitement de connaissances.
   - Ils communiquent de manière évoluée (ce qui est en fait une conséquence de leurs processus cognitifs évolués).
   - Ils sont plus proches des humains que des fourmis.

Mais il est tout à fait possible d'avoir des agents qui cummulent des fonctionnalités de types réactives et des fonctionnalités de type cognitives. En fait, si un agent accomplit plusieurs tâches, il y a fort à parier que certaines tâches seront traitées de manière réactive et les autres de manière cognitive. Les agents permettent, par exemple, de résoudre des problèmes selon un mécanisme d'*écorésolution*.
C'est à dire que toutes les règles de comportement des agents sont locales. Le système est traité au niveau microscopique et non macroscopique. Ce peut-être une définition de ce qu'est un agent, *un agent est une entité qui travaille à l'échelle microscopique et sont travail ne prend de sens que considéré avec le travail des autres agents au niveau macroscopique (swarm intelligence).*

La communauté Internet
Puis la communauté Internet s'est mise à utiliser ce terme que l'on commençait à lire un peut partout (1994-1996) dans les travaux des scientifiques (travaux qui circulaient aussi sur l'Internet). Mais cette communauté si elle était charmée par le terme, le comprenait sous une acceptation un peu différente.
Et donc depuis approximativement 1996, pour la communauté Internet, un agent est plutôt un *assistant informatique*, un agent personnel, appelé aussi **Software Agent**. Cet agent assiste l'utilisateur dans son travail.
Cependant le sofware agent se conçoit d'abord seul. Il peut y en avoir plusieurs dans un système, ou pour l'accomplissement d'un processus, mais ils peuvent très bien fonctionner indépendemment les uns des autres.

Types des agents du projet
Nous proposons des agents des deux types traités plus haut.
C'est à dire aussi bien des agents pour SMA, qui agissent au niveau microscopique, que des Software Agents, qui agissent au niveau macroscopique.

Les agents SMA communiquent entre eux et arrivent à des décisions, des compromis, au niveau microscopique des processus de décision ou de traitement des tâches, et agissent par coalition ([45]

[44]). Devant les fortes contraintes de temps qui nous ont été imposées, nous nous sommes concentrés sur les agents de prise de rendez-vous. Mais nous nous étions aussi proposés au départ de réaliser des agents d'échanges d'information, pratiquant une sorte de "marchandage" d'informations.

Les Software Agents accomplissent quant à eux des tâches simples comme la recherche d'informations ou la maintenance. Ces derniers n'ont aucune raison particulière de travailler en coopération.

Chapitre 4

Architecture du Système d'Information

4.1 Infrastructure

L'infractructure de notre système au niveau des relations entre agents peut être comparée à la méthode de coalition([44]) pour l'accomplissement de tâches et à RETSINA ([45]) pour la décomposition en trois classes d'agents (agents d'interface, agents de tâche, agent d'information) ou bien InfoSleuth ([38]).

4.2 Agents dans un système de GroupWare

Pour représenter les utilisateurs dans un système de GroupWare, il faut se représenter les différentes entités auxquelles les personnes physiques appartiennent, et appliquer une métaphore au système. [49] distingue quatre types de groupes:

- les équipes : des groupes de personnes qui se connaissent, se font confiance, et qui travaillent pour un même but. Chaque personne dépend énormément des autres et doit savoir ce que fait chaque autre personne.
- les organisations : groupe organisé par des structures et modèles de pouvoir. Cela regroupe de grands nombres de personnes qui ne se connaissent pas forcémment, et qui ne partagent pas les mêmes buts. Chaque personne travaille individuellement et peut être engagé dans plusieurs tâches en même temps.
- les groupes d'interaction sociale : personnes se connaissant, et n'ayant pas les mêmes buts. Le média de communication habituel est la conversation (papier, email...)
- les groupes amorphes : gourpes de personnes qui ne se connaissent pas, font parti de grands groupes, ne partagent pas les mêmes buts, ne se font pas confiance... Ils partagent un espace d'information.

On peut appliquer à chaque groupe des types d'agents différents. Il est clair qu'il faut prévoir la gestion des accès aux données et ressources dans ces différents groupes en interaction. Trois types d'agents sont décrits:

- *Pushers* qui envoient des informations ([26])
- Synchronisateurs de données
- Médiateurs d'accès

Agents que l'on peut rapprocher dans le même ordre que ceux décrits dans [45] et utilisés dans notre système d'agents :

- Agents d'Interface
- Agent de Tâche
- Agent d'Information

L'agent d'Interface (Interface Agent) interagit avec l'utilisateur, reçoit ses spécifications et renvoit les résulats. Il acquiert, modélise et utilise les préférences de l'utilisateur.

L'Agent de Tâche (Task Agent) formule les plans et les mènes à bien. Il a :

- la connaissance du domaine de la tâche, et avec d'autres Task Agents ou Agents d'Information, est capable d'accomplir différentes parties de tâches.
- des stratégies pour résoudre les conflits et fusionner l'information reçue par les Agents d'Informations.
- la capacité de décomposer les plans et coopère avec les Agents de Tâche appropriés, ou Agents d'Informations pour l'exécution d'un plan, la surveillance et la composition d'un résultat.

L'agent d'Information fournit un accès intelligent à une source d'information hétérogène de sources d'informations. Il a les modèles des sources d'informations et les stratégies pour leur sélection. Il est actif car il surveille les sources d'informations, et fournit l'information.

4.3 Déplacement des données

Déplacer les données avec les agents est un phénomène coûteux à prendre en compte. Les machines sont un support fini, et leurs capacités ne sont donc pas infinies, en ce qui concerne le stockage et le transport de données. Quelques règles simples ([42]) peuvent être prises en compte concernant ces déplacements:

Pour un environnement de serveurs appelés par la suite 'SERVERS', connectés entre eux par un réseau. Les données contenues sur chaque serveur sont représentée par des grappes d'informations (Data Sets DS). On peut représenter l'usage qui est fait des données par $SERVERS \times DS \longmapsto R^+$. C'est une fonction qui associe à chaque serveur et grappe de données, le nombre de documents de cette grappe de de données qui seront demandés par des clients dans la zone géographique de ce serveur, pendant une période de temps donnée (semaine, mois...). De plus $storage\_cost$ représente le coût pour stoquer localement une unité de données pour une période de temps, et la fonction $dataset\_size$ spécifie la taille de chaque ensemble de données en unité de données. $storage\_cost$ et $answer\_cost$ sont communs à tous les serveurs.

Définition: *Le profit que le serveur* s *s'attend à obtenir en stoquant la grappe de données* ds *localement pour une période* te *temps donnée est:*

$$V_s(ds) = -storage\_cost * dataset\_size(ds) + \sum_{s' \epsilon SERVERS} (usage(s', ds) * (query\_price - distance(s, s') * answer\_cost))$$

Définition: *Le profit que le serveur* s *s'attend à obtenir d'une grappe d'information* ds *située à la position* loc *du temps 0 à N, connaissant* V_s *est:*

$$P_s(ds, loc) = \begin{cases} \sum_{t=0}^{N} \frac{V_s(ds)}{(1+r)^t} & loc = s \\ 0 & otherwise \end{cases}$$

où r *est le taux d'intérêt, et N est le nombre de périodes durant lequel l'environnement existe. Si l'environnement est considéré existant indéfiniment, alors* $N = \infty$. *Dans ce cas:*

$$P_s(ds, loc) = \frac{V_s(ds) * (1 + r)}{r}$$

A partir de là, on peut utiliser cette fonction de coût pour savoir si les agents doivent se déplacer, déplacer l'information, ou s'il vaut mieux la laisser sur place.

Il s'ensuit un protocole de Commerce/Demande (Trading/Bidding), avec élection de la part du contracteur (contractor):

Définition:

$$winner(ds) = \begin{cases} argmax_{bidder \epsilon SERVERS} \left(price\_suggested(bidder, ds) - move\_cost(ds, bidder)\right) & move(ds) = true \\ none & otherwise \end{cases}$$

S'il y a plus d'un proposant (bidder), avec la même valeur maximale de $price\_suggested(bidder, ds) - move\_cost(ds, bidder)$, alors le contracteur sélectionnera l'un d'eux (arbitrairement) pour être le gagnant ($winner$), et les autres seront considérés comme étant les proposants avec la seconde meilleure offre.
D'autres applications de cette stratégie sont décrites dans [42].

4.4 Relations Client / Serveur

Notre système d'information est composée de plusieurs serveurs tournant sous des systèmes d'exploitation différents. Des agents mobiles se déplacent de serveurs en serveurs.

Un scénario d'utilisation du point de vue de l'utilisateur comporte 5 phases (Fig. 3.2 page 22).

1. Demande de connexion.
2. Authentification par le serveur principal à l'aide de la BdD utilisateurs.
3. Transfert des agents et des préférences de l'utilisateur vers le serveur le plus adapté (le plus proche et le moins chargé en terme de charge processeur).
4. Début d'une séance de travail. Communications entre l'utilisateur et les agents.
5. Les agents se mettent au travail. Ils se déplacent de serveurs en serveurs. Ils peuvent communiquer à tout moment depuis n'importe où avec leur propriétaire. Les communications transitent alors par le serveur sur lequel travaille l'utilisateur.

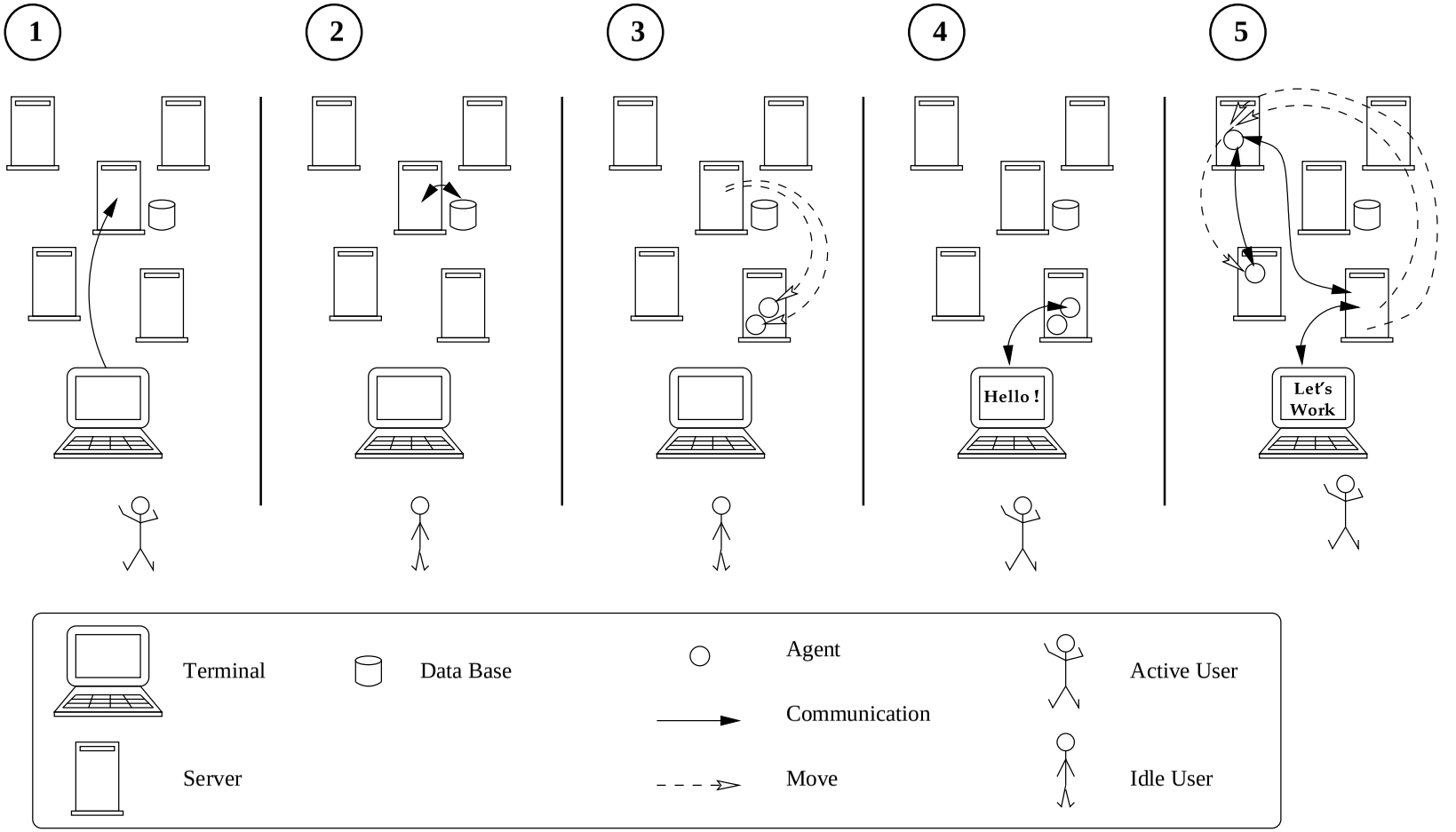


FIG. 4.1 – *Scénario d'utilisation*

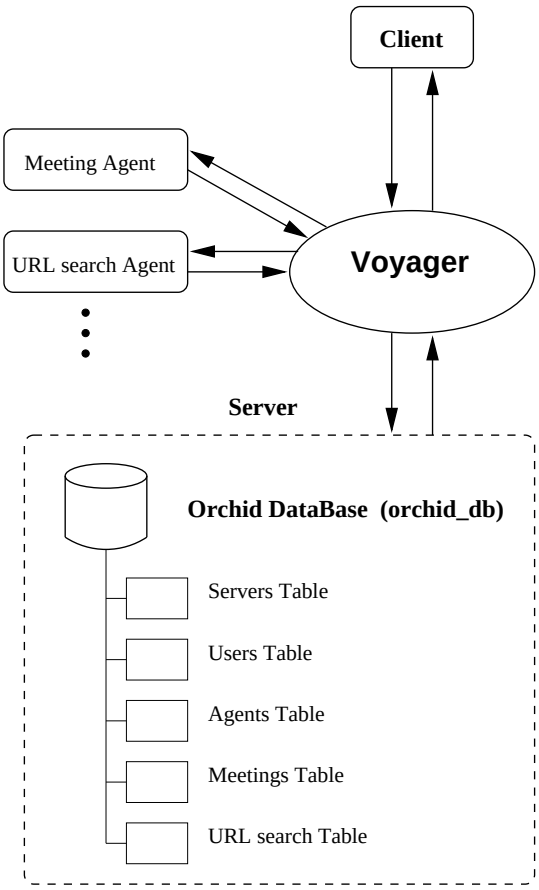


FIG. 4.2 – *Architecture des relations Client / Serveur*

4.5 Tables de la Base de Données Orchid

Table servers

name	url	availability	master
i5.esiea.fr	192.168.1.105	10	t
n7.esiea.fr	192.168.1.170	10	f
nef.esiea.fr	192.168.1.103	4	f

L'url de chaque serveur est unique.
Plus l'*availability* d'un serveur est élevée, plus ce serveur est apte à recevoir des connexions, et moins sa charge est élevée. Une *availability* de 0 signifie que le serveur ne peut plus accepter de nouvelle connexion. Une valeur de -1 signifie que le serveur est non-connecté.
Le champ *master* indique quel est le serveur principal.

Table users

name	password	info	profile	online
philippe.benhamou	a23q	ONERA		t
antoine.souliez	q$;s	ESIEA		f
ma.darche	8;az	ESIEA		t

Le nom de chaque utilisateur est unique.
Il s'écrit sous la forme <prénom>.<nom>.
Le champs *online* a la valeur TRUE si l'utilisateur est connecté et FALSE s'il ne l'est pas.

Table agents

name	owner	home	currentlocation	frozen
ma__philippe.benhamou	philippe.benhamou	192.168.1.105	192.168.1.170	t
ma__antoine.souliez	antoine.souliez	192.168.1.105	192.168.1.170	f
ma__ma.darche	ma.darche	192.168.1.105	192.168.1.105	f

Le nom de chaque agent est unique.
Il s'écrit sous la forme <type d'agent>__<propriétaire de l'agent>.

Table meetings

user	subject	note	initiator
philippe.benhamou	réunion tped	définition du terme agent	ma.darche

date	timeofday	duration
1/10/1998	1430	4

participants	frozen
ma.darche emmanuel.pierre frédéric.salvy antoine.souliez mo.varroy	f

Le champs *frozen* indique s'il est encore possible que la réunion puisse être modifiée.

Table urlsearch

user	subject	list	date

Chapitre 5

Graphe d'enchaînement des écrans

FIG. 5.1 –

FIG. 5.2 –

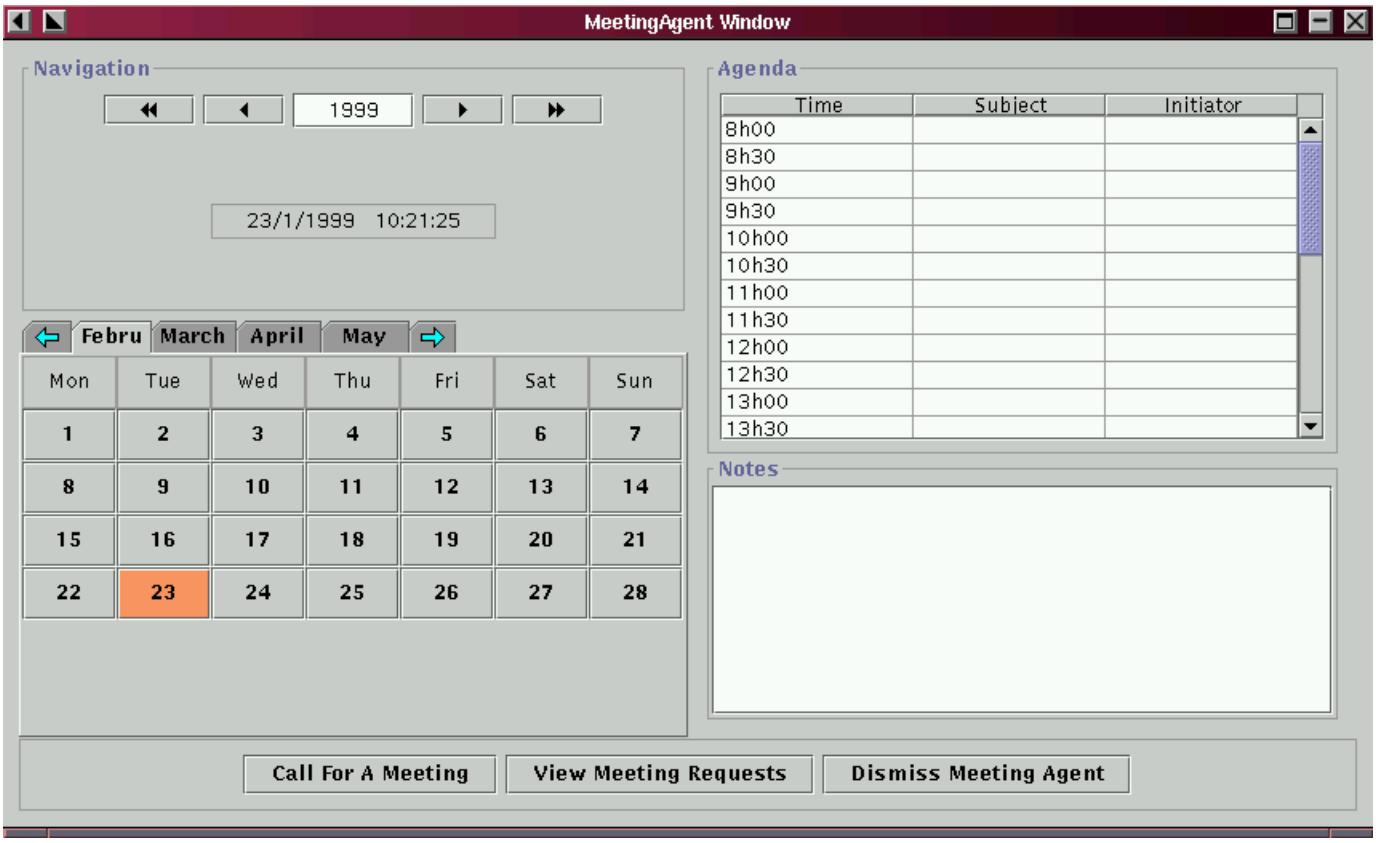

MeetingAgent Window
Navigation
1999
23/1/1999 10:21:25
Febru
March
April
May
Mon
Tue
Wed
Thu
Fri
Sat
Sun
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
Agenda
Time
Subject
Initiator
8h00
8h30
9h00
9h30
10h00
10h30
11h00
11h30
12h00
12h30
13h00
13h30
Notes
Call For A Meeting
View Meeting Requests
Dismiss Meeting Agent

FIG. 5.3 –

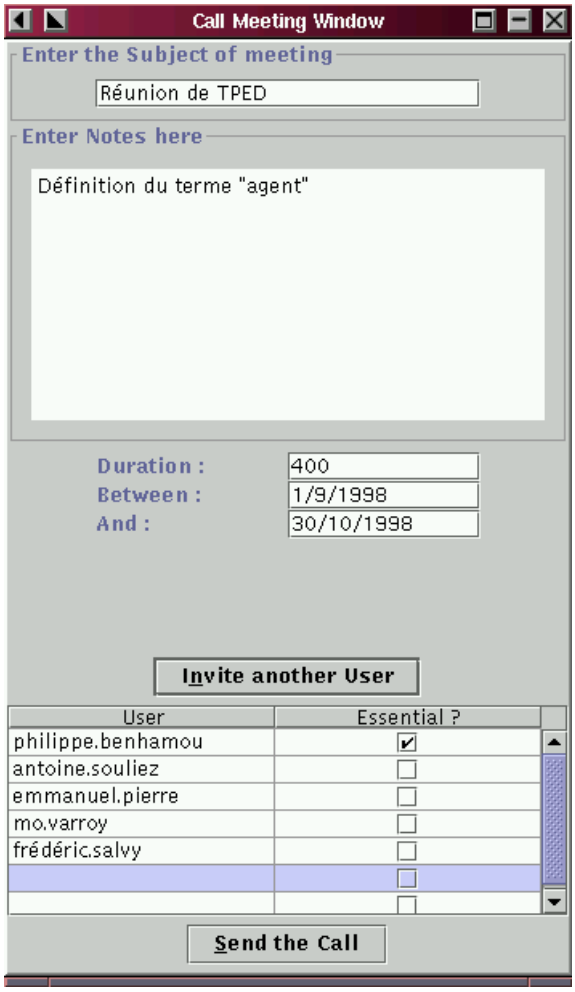

Call Meeting Window
Enter the Subject of meeting
Réunion de TPED
Enter Notes here
Définition du terme "agent"
Duration :
400
Between :
1/9/1998
And :
30/10/1998
Invite another User
User
Essential ?
philippe.benhamou
antoine.souliez
emmanuel.pierre
mo.varroy
frédéric.salvy
Send the Call

FIG. 5.4 –

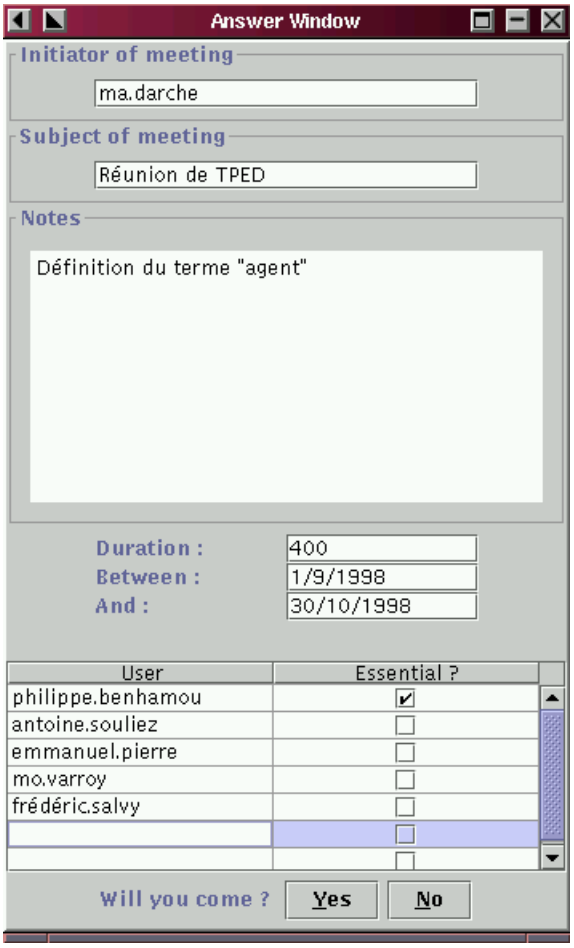

Answer Window
Initiator of meeting
ma.darche
Subject of meeting
Réunion de TPED
Notes
Définition du terme "agent"
Duration :
400
Between :
1/9/1998
And :
30/10/1998
User
Essential ?
philippe.benhamou
antoine.souliez
emmanuel.pierre
mo.varroy
frédéric.salvy
Will you come ?
Yes
No

FIG. 5.5 –

Chapitre 6

La prise de Rendez-Vous

6.1 l'IAD dans la prise de Rendez-Vous

Pourquoi utiliser l'IAD dans une prise de Rendez-Vous?

- il faut un agent par participant (pour respecter la vie privée).
- cela permet à l'utilisateur de modifier l'algorithme de l'agent et de le personnaliser plutot que de modifier le serveur central.
- la distribution des agents évite les goulots d'étranglement.
- les calculs sont distribués sur toutes les machines.
- les agents permettent la représentation abstraite d'un assistant personnel.
- la redondance des serveurs permet une meilleure tolérance aux pannes.

Il faut prendre aussi en compte le pire des cas, formulé par le "cauchemard des secrétaires" ([28]). L'exemple de base est composé de quatres personnes accompagnées de beaucoup de contraintes sur la solutions possible. On peut estimer facilement que le problème est très complexe.

D'où quelques remarques sur le système ([43]):

- tous les calendriers ne peuvent être stoqués sur le même serveur (surtout si l'on travaille avec des personnes extérieures à l'entreprise (voir le détail des groupes de travail dans la partie GroupWare et [49])). D'ailleurs, aucun agent n'a besoin d'accéder aux détails du calendrier d'un autre agent.
- si on a N personnes attendues pour la réunion, on a $O(N^2)$ chemins de communication entre agents.
- si l'initiateur prend en charge la gestion, on en a seulement $O(N)$

Il convient aussi de prendre encore deux facteurs en compte par personne:

- l'importance du Rendez-Vous (noté de 0 à 100).
- l'importance de la personne dans le Rendez-Vous.

6.2 La méthodologie

Ce projet nécessite une grande méthodologie car il est composé de deux entités.
Ces 2 entités sont l'**architecture technique**, c'est à dire l'architecture client/serveurs du système, et les **agents**.

Nous avons utilisé la méthodologie pour les Systèmes Multi-Agents de (Moulin 1995 [36]).
Nous avons donc suivi les 2 étapes *Analyse* puis *Design*, en abordant les sous-étapes qui nous sont utiles.

6.3 L'écorésolution

Nous sommes parti de l'idée d'écorésolution (Ferber 1990 [30]) chaque personne nécessaire pour un Rendez-Vous est représentée par des agents qui interagissent tous entre eux. Mais si de nombreuses personnes doivent être présentes, nécessite une puissance de calcul phénoménale.
Actuellement, ceci est faisable à partir d'architectures massivement parallèles comme les BeoWulf ([5], [6]), mais ne présente pas en soi une conception IAD à partir d'agents mobiles.

6.4 L'algorithme

En ce qui concerne l'algorithme nous nous sommes basés sur celui proposé pour les Systèmes Multi-Agents [36].
Nous l'avons adapté de manière à ce qu'il prenne en compte l'architecture technique de notre système.
Cette adaptation fait passer un algorithme d'une dizaine d'étapes à une vingtaine.
Nous nous sommes basés sur les agents et leur organisation entre eux, une autre approche basée sur la métaphore des fourmis [46] en fournit une autre.

6.5 Analyse

6.5.1 Description du scenario

Nous avons choisi qu'il y ait un **agent pour le carnet de rendez-vous (RDV) de chaque utilisateur**.

> Il s'agit bien ici d'IAD car il y a des négociations entre les agents des différents utilisateurs. Mais il y aurait eu de l'IAD de manière encore plus directe s'il y avait eu un agent pour chaque plage de RDV d'un utilisateur. Le système de résolution aurait alors été de l'**écorésolution** (Ferber 1990 [30]).
> Au début de nos entretiens avec P. Benhamou nous avons été séduits par l'écorésolution. Nous avons cependant dû y renoncer face aux capacité de calcul qu'une telle approche demande. En effet il aurait fallu générer **un processus (Thread) par agent**, le langage de programmation du système étant Java. Et quand on veut des plages de rendez-vous d'une demi-heure, que les RDVs peuvent avoir lieu entre 8h et 18h, que les RDVs peuvent être pris plusieurs semaines à l'avance (par exemple 2) et qu'enfin les entreprises pouvant être intéressées par notre produit peuvent compter plusieurs centaines de personnes (par exemple 200), on arrive à l'estimation suivante : 12000 Threads !!!
> Sachant qu'un Thread occupe 10ko de mémoire vive et 2% de la charge CPU d'un Intel Pentium 200Mhz, il faudrait une machine équipée de 10 ce ces processeurs ainsi que de 400Mo de mémoire vive.
> Ce type de résolution requiert alors l'utilisation d'un cluster de plusieurs machines. C'est désormais possible avec la distribution de Linux Beowulf qui est développée par la NASA [5].
> L'écorésolution pose aussi un problème de convergence du système à résoudre. C'est à dire qu'on est jamais sûr que le problème puisse être résolu à l'aide de cette méthode. Cette méthode de résolution serait très intéressante à analyser de manière à déterminer si les états du système présentent des cycles, s'il y a des états stationnaires différents des états stables. L'analyse théorique de cette méthode est aussi intéressante que difficile et sort du cadre de cette étude plus orientée sur l'implémentation des agents.

L'initiateur
On appelle l'utilisateur qui demande le RDV l'initiateur.
Les invités
On appelle les utilisateurs qui sont conviés à un RDV les invités.
Les participants
On appelle les participants, les invités ayant exprimé leur accord de se rendre au RDV.

1. Phase de formulation
Il s'agit de la construction de la demande par l'initiateur. Ce dernier peut préciser quelles sont les invités essentiels pour la réunion. La formulation comprend:

- L'intitulé de la réunion.
- La durée prévue de la réunion.
- La période pendant laquelle la réunion peut avoir lieu (date début de période et date de fin de période).
- La liste des invités.
- Et (facultatif) les invités indispensables à la réunion.

Pour ce qui suit, on se reportera à (Fig. 5.1 page **??**).
La partie haute de la figure représente l'aspect logique du scénario. La partie basse correspond quant à elle à l'aspect physique du scénario. L'architecture Client/Serveur du système étant relativement complexe, les deux aspects doivent être considérés en même temps.

2. Phase de demande de participation [Etapes 1 et 2 de la figure]
Chaque invité reçoit la demande de participation. Cette demande comprend tous les renseignements relatifs à la réunion (qui vient, qui est *essentiel*, etc.).
Il doit répondre Oui (je suis intéressé par la réunion) ou Non (je ne suis pas intéressé par la réunion/ ne veux pas venir). L'absence de réponse sous 24h est pris pour une réponse négative.
Il n'est en effet pas utile de poser des problèmes de prise de RDV à cause de personnes qui de toute façon ne viendront pas à la réunion.
A l'issue de cette phase on obtient donc la liste des participants.
Si les invités décrits comme *essentiels* pour la réunion fournissent une réponse négative, ou ne répondent pas avant 24h, alors la réunion est ajournée.

3. Phase de détermination des contraintes et des rôles [Etape 3 de la figure]
Comme expliqué dans (Moulin 1995 [36]) les agents déterminent le Most-Constrained Participant (MCP) (c'est à dire le participant ayant le plus de contraintes), ensuite le Second-Most-Constrained Participant (2MCP) (c'est à dire le second participant ayant le plus de contraintes), etc.

Pendant cette phase, chaque agent calcule son degré de contrainte et l'envoie à tous les autres agents. Chaque agent reçoit donc le degré de contrainte de chacun des autres agents.
Cela permet à chaque agent de se constituer une base de connaissances qui comporte la liste des agents participants classés par ordre du MCP vers le nMCP (c'est à dire du plus contraint au moins contraint).

4. Phase de négociation [Etape 3 de la figure]
Les négociations partent du MCP vers le 2MCP, puis vers le 3MCP, etc. , de manière à optimiser le processus et à éviter les vérifications inutiles.
Plus précisemment, le MCP propose au 2MCP sa première plage libre pour le RDV.
Si cela convient au 2MCP, il propage la proposition, c'est à dire qu'il la transmet au 3MCP. Si cela ne lui convient pas, il demande au MCP une autre proposition de plage.
Et ainsi de suite.

5. Fin des négociations au niveau sous-groupe [Etape 4 de la figure]
La fin des négociations est atteinte :

1. si le MCP ne peut plus proposer de plages au 2MCP. Dans ce cas, le MCP renvoit un message à l'initiateur lui annonçant que le RDV ne peut avoir lieu dans la plage demandée. La réunion est alors ajournée.
2. lorsque le nMCP (c'est à dire l'agent le moins contraint) reçoit une proposition de RDV de la part du (n-1)MCP qui lui convient. Dans ce dernier cas, le nMCP rejoint l'agent initiateur.

6. Fin globale des négociations [Etape 5 de la figure]
Là aussi il y a deux possibilités :

1. Si un MCP d'un sous-groupe n'a pas pu déterminer de plage pour le RDV, il rejoint l'agent de l'initiateur et le lui signale. La réunion est alors ajournée.
2. Si tous les nMCP de tous les sous-groupes rejoignent l'agent de l'initiateur avec une plage pour le RDV, ce dernier calcule alors l'intersection des plages. Le RDV est alors décidé pour avoir lieu à l'intersection des plages, et la date et l'heure sont envoyées aux agents des participants.

Relachement des contraintes
Pour que des compromis soient plus facilement atteints, on peut perfectionner les agents de manière à leur permettre de relâcher les contraintes sur plages horaires des agendas des utilisateurs.
Pour que cela soit possible, il faut gérer les contraintes de la manière suivante:

1. L'agent de l'initiateur demande tout d'abord aux autres agents de diminuer les contraintes sur certaines de leurs plages horaires. C'est à dire, que si par exemple une pause déjeuner est notée comme "gênante", dans l'agenda d'un utilisateur, elle est renotée “disponible” le temps de la nouvelle négociation. Les négociations reprennent alors en ayant plus de chances d'aboutir.
   Il en va de même pour d'autres RDVs qu'un participant a noté comme “gênants” et non comme “indéplaçables”.
2. Si le RDV n'est toujours pas possible certains participants non essentiels sont écartés de la réunion de manière à maximiser le nombre de personnes présentes.

B

Blocage de transactions
Pendant une demande de RDV de la part d'un utilisateur, tous les agents des "invités" n'ont plus le droit d'être invités pour un autre RDV, jusqu'à ce que la transaction aboutisse (de manière positive ou non).

Cela n'empêche nullement plusieurs personnes de prendre des RDV au même instant sur le système. Cela empêche à une même personne d'être demandée pour plusieurs RDVs en même temps. Les agents permettent de faire des analogies avec leurs homologues humains. Il est en effet très dur pour un humain de prendre 2 RDVs en même temps.

Annulation de RDVs
Les RDVs décidés par les agents peuvent être résilliés à tout moment par l'utilisateur.
Dans ce cas, l'agent de prise de RDV de l'utilisateur envoie cette information à tous les agents des autres participants, de manière à mettre à jour leurs données sur la réunion (donc entres autres qui y participe). Si l'utilisateur est essentiel à la réunion un message est envoyé à tous les agents des participants pour leur dire que la réunion est annulée.

Déplacement de RDVs
Un participant peut demander à posteriori qu'un RDV soit déplacé. Les RDVs ne sont plus déplaçables après une certaine période avant la date du RDV. Autrement dit, si la date d'un RDV est très rapprochée de l'instant présent, on interdira que ce RDV puisse être modifé, de manière à ne pas gêner les autres participants. Donc, passé un certain temps, tous les RDVs deviennent indéplaçables (mais ils sont toujours annulables).
Ce point n'a pas été développé plus avant.

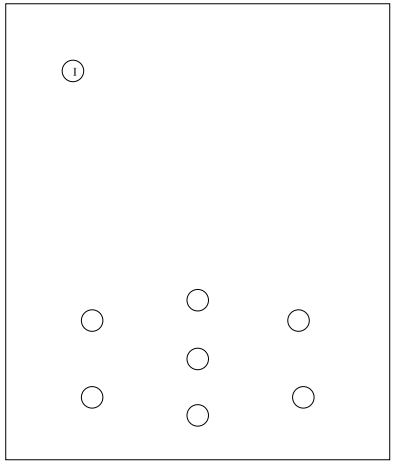


Fig. 1: Situation initiale

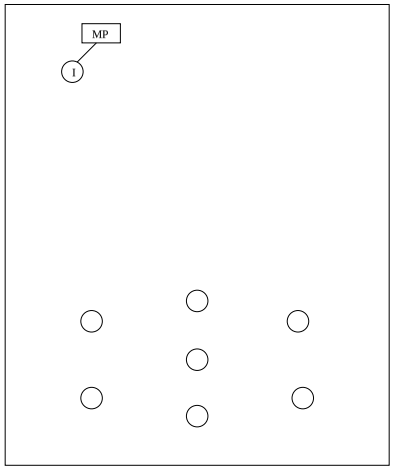


Fig. 2: Création du *Meeting Proposal*

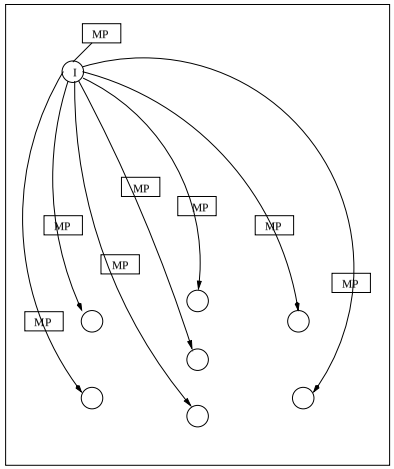


Fig. 3: L'Initiateur distribue le MP aux invités

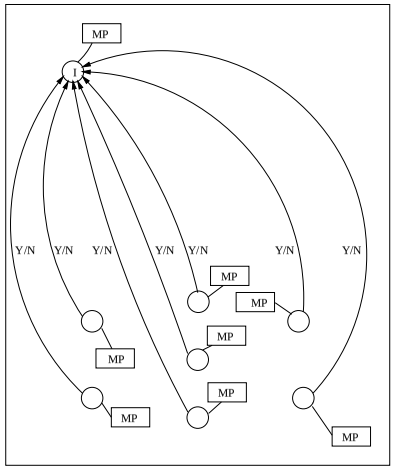


Fig. 4: Réponse des invités

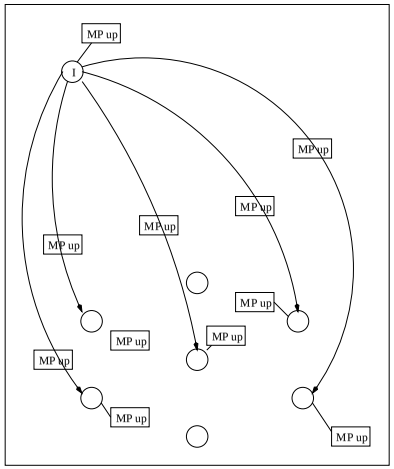


Fig. 5: Mise à jour du MP avec les participants

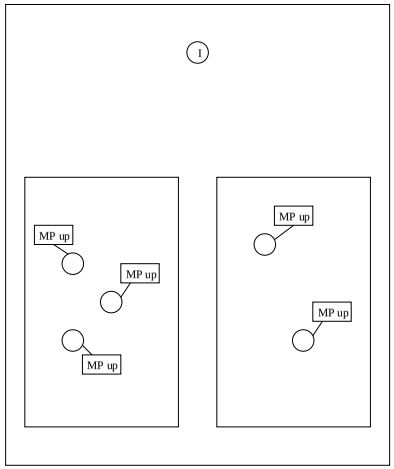


Fig. 6: L'Initiateur répartit les agents entre les différents serveurs, après avoir consulté la disponibilité des serveurs, suivants un algorithme donné

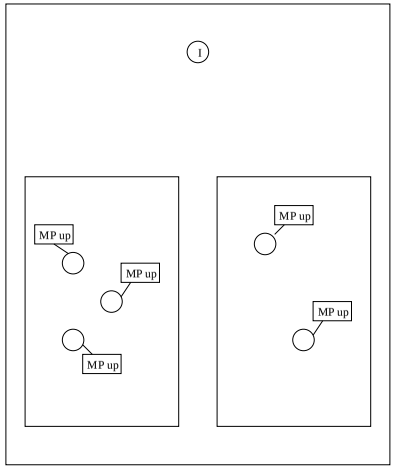


Fig. 7: Création de sous-espaces pour faciliter la communication

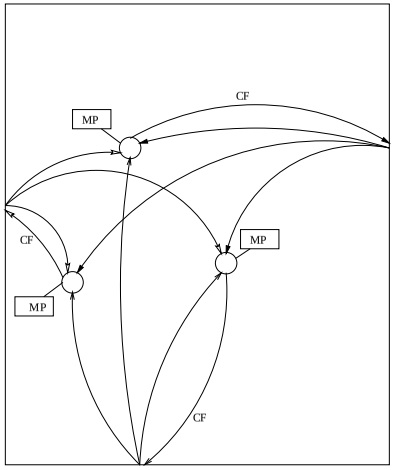


Fig. 8: Calcul et envoi des *Contraint Factor* aux autres agents via le sous-espace

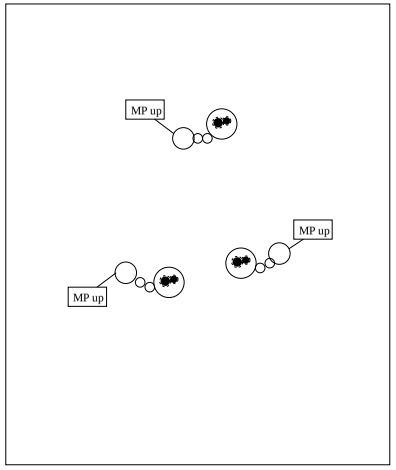


Fig. 9: Tri des CF pour mettre dans l'ordre les MCP

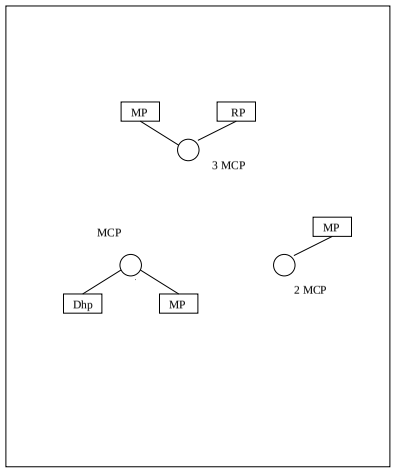


Fig. 10: Création d'un *Data Hour Proposal* pour le MCP, et d'un *Range Proposal* pour le nMCP

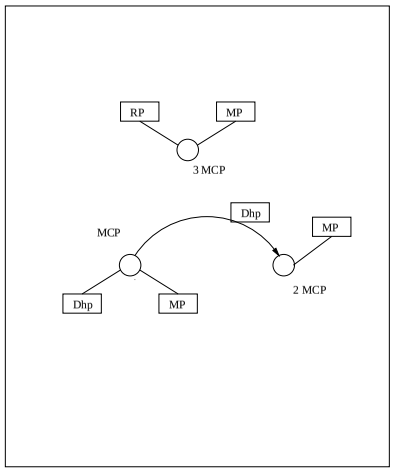


Fig. 11: Envoi d'un Dhp vers le prochain MCP

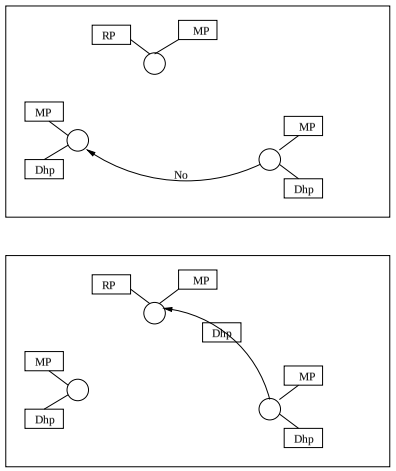
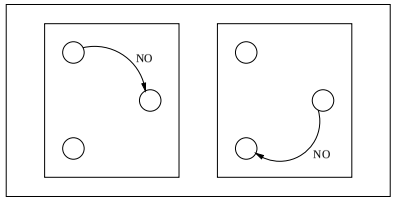


Fig. 12a: Le 2MCP n'accepte pas le Dhp
Fig. 12b: Ce Dhp convient au 2MCP qui la passe au suivant



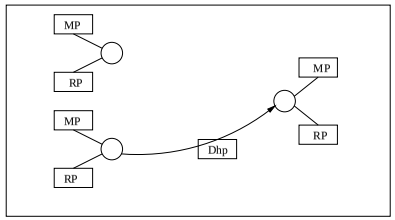


Fig. 13a: Le 3MCP n'accepte pas le Dhp et le refus est propagé
Fig. 13b: Le MCP propose un autre Dhp au MCP suivant

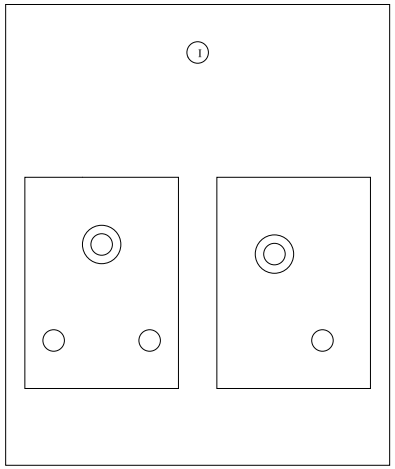


Fig. 14: Le nMCP remplit le RP avec les Dhp approuvés par tous ceux du groupe. Tant que le MCP peut proposer des Dhp inférieurs à la date limite, on retourne a la figure 11

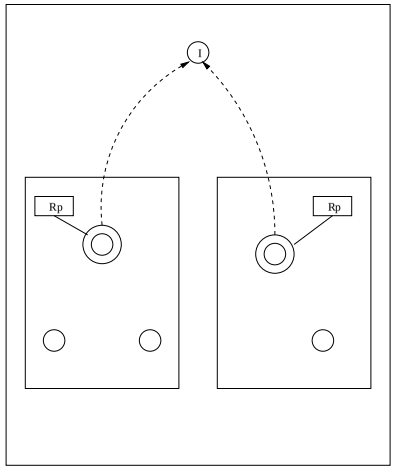


Fig. 15: Déplacement des nMCP de chaque groupe vers l'Initiateur

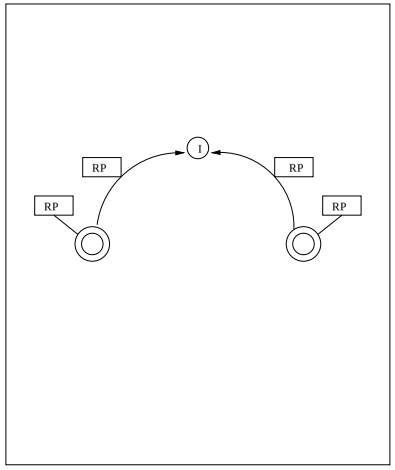


Fig. 16: Communication des RP de chaque groupe à l'Initiateur

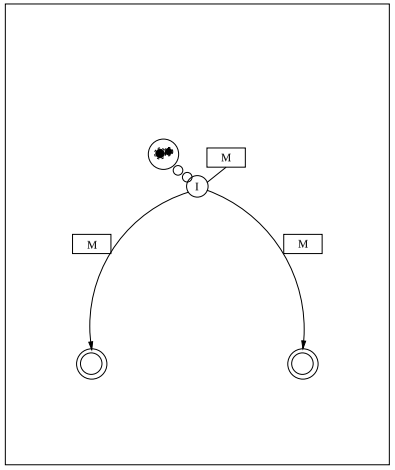


Fig. 17: Résolution par l'Initiateur pour trouver un rendez-vous et communication du *Meeting* (si trouvé) aux nMCP

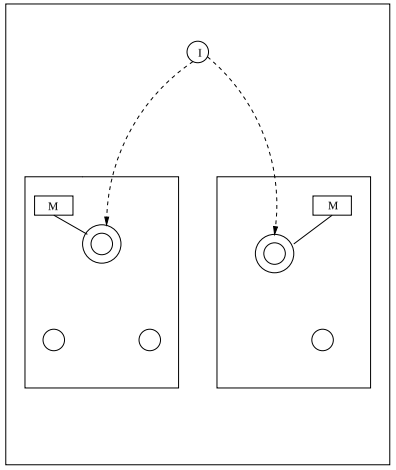


Fig. 18: Retour des nMCP dans leur groupe avec le *Meeting*

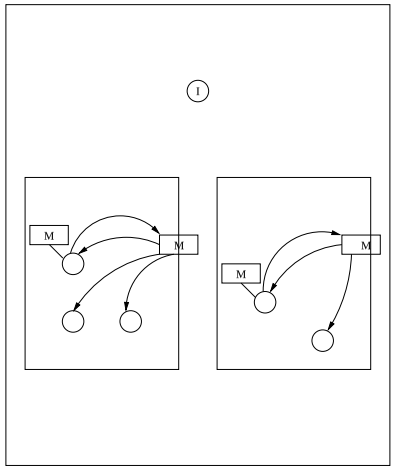


Fig. 19: Le nMCP communique le M aux autres via le sous-espace

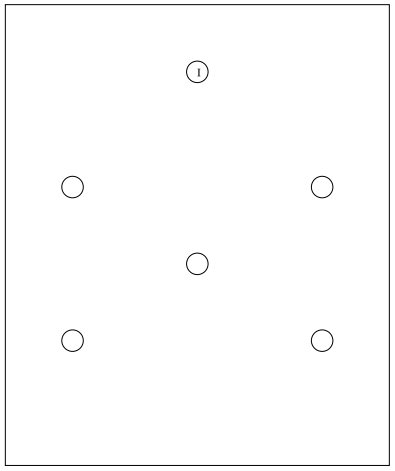


Fig. 20: Chacun retourne chez lui

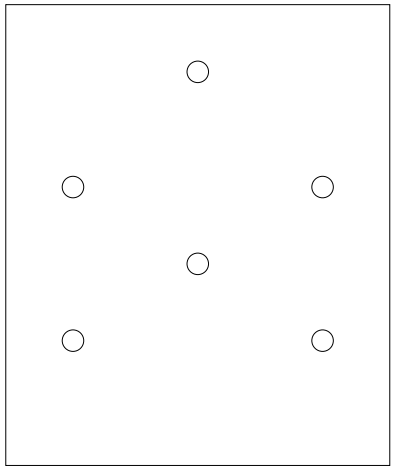


Fig. 21: Mise à jour des emplois du temps

Aspect physique de l'architecture du système d'information
Voir (Fig. 5.1 page **??**), partie basse

1. L'agent de l'initiateur envoie une proposition de RDV à tous les autres agents.
2. Tous les agents des invités lui renvoient (ou non) leur réponse.
3. Les agents des participants se répartissent sur les différents serveurs de manière à former des sous-groupes de travail.
4. A la fin des négociations au niveau des sous-groupes, un agent par sous-groupe rejoint l'agent de l'initiateur pour la détermination finale de la plage du RDV.
5. Les agents qui se sont déplacés retournent sur les serveurs où ils se trouvaient avant la demande de prise de RDV. L'agent de l'initiateur communique alors à tous les participants la plage déterminée pour le RDV.

6.5.2 Description Fonctionnelle des rôles

Méthode de Coalition
Par définition, une coalition a lieu quand un groupe d'agents se décide à coopérer pour accomplir une tâche commune. Ce groupe n'effectue qu'une tâche à la fois, mais un agent peut appartenir à plus d'une coalition ([45] [44]).

L'initiateur
Chaque fois que l'initiateur reçoit la liste des participants, son agent établit un *découpage des participants en sous-groupes de travail.* Tous les agents d'un même sous-groupe se déplacent alors vers un même serveur sur lequel ils négocient. Ceci est fait pour diminuer la charge de travail sur chaque machine. C'est l'agent de l'initiateur qui calcule sur combien de serveurs doivent être répartis les agents des participants.

Les participants
Si il n'y a qu'un seul participant, la négociation s'effectue entre l'agent de l'initiateur et celui du participant.
Si il y a un groupe de participants, la négociation a d'abord lieu entre les participants du même sous-groupe.

6.6 Design

6.6.1 Modélisation des Objects

Il y a :

- MeetingProposal
- DateHourProposal
- RangeProposal
- Meeting
- MeetingAgent
- SAAgent

6.6.2 Modélisation des Agents (construction des agents)

On définit un type générique d'agent : le MeetingAgent.
Il dispose d'un ensemble de méthodes.

6.6.3 Modélisation des Conversations des Agents

Pour modéliser les conversions des agents entre eux, nous nous appuyons sur les spécifications du Knowledge Query and Manipulation Language (KQML) développé par le DARPA Knowledge Sharing Effort, sur la base des actes du langage.

Les spécifications du KQML datent de 1996, et peuvent être trouvées sur le site dédié au langage [15].
Il s'agit d'un langage de type *Performatif*, c'est à dire qu'il sert à interroger ou donner des ordres ("Communiquer c'est agir" A.Drogoul "Toute communication peut être négociée" C.Hewitt).

Cela implique que:

- tout agent est vu comme gérant une base de données.
- KQML offre des primitives pour manipuler ces primitives au cours d'un dialogue.

L'objectif étant de permettre l'inter-opérabilité entre agents hétérogènes sur Internet (software agents [33]).

On peut citer [48] qui utilise la même architecture et qui utilise la métaphore de courrier envoyé de groupes d'agents à d'autres, comme un mélange de BlackBoard et de Speech Acts.

Un très bon exemple de mise en pratique de la transaction par KQML est InfoSleuth (Nodin M., Unruh A. 1997 [38]). Nous avons un peu adapté les spécifications de manière à traiter notre résolution plus facilement.

On prend Meeting comme *ontology*.

Cette surcouche n'est pas actuellement utilisée dans notre projet, quoique facilement insérable grâce à la technologie Objet.

Organisation du travail

Avancement du projet

Début Juin 1998

- Début du TPED

Mi Juin 1998

- Première réunion avec M. Benhamou

Fin Juin 1998

- Soutenance du projet avec messieurs Benhamou, Ramond et Erra.
- Debriefing avec M. Benhamou.

Septembre 1998

- Programmation: Début de la création de l'interface pour le poste client.

Réunion du 1er Octobre 1998 avec M. Benhamou

- Définitions des différentes acceptations du terme "agent".
- Choix des types d'agent pour le projet.
- Choix de l'organisation des agents et de leur manière de communiquer.

Email du 3 Octobre 1998 de M. Benhamou

- Réception d'un rapport sur un système multi-agents présentant des scénari de communication.

Réunion du 8 Octobre 1998

- Mise au point des scenari de communication des agents sur les conseils de M. Benhamou.

Réunion du 10 Octobre 1998

- Rédaction finale et saisie informatique des scenari en vue de les envoyer par Email à M. Benhamou.

Semaine du 12 au 16 Octobre 1998

- La majorité de l'équipe suit le cours "Agents logiciels intelligents" de Pavlos MORAÏTIS (moraitis@lamsade.dauphine.fr). Le projet subit quelques modifications suite aux nouvelles données apportées par le cours.

Réunion du 31 Octobre 1998

- Modification au niveau de la prise de RDV : les participants ne sont plus répartis selon une structure hiérarchique. On décide plus simplement de les diviser en sous-groupes de travail.
- Envoi par Email à M. Benhamou de la dernière version du rapport comprenant les scenari de prise de RDV.

10 Novembre 1998

- Les serveurs I5 (dual boot Linux/NT) et NEF (Linux) sont finis d'être configurés. Sont installés JDK, SWING, PostgreSQL, Voyager.

Réunion du 30 Novembre 1998

- Discussions sur l'implémentation des agents grâce à Voyager: création, freezing et mort des agents.
- Décision d'utiliser un fichier "servers_list.cfg" au niveau du client.

Début Décembre 1998

- Définition de *IMeetingAgent*, l'interface des agents de prise de RDV.
- Définition des objets que les agents de prise de RDV manipulent: *Meeting*, *MeetingProposal*, *DateHourProposal* et *RangeProposal*.
- Programmation: réalisation des objets et des différents serveurs.

Fin Décembre 1998

- Organisation de toutes les classes en *packages*.
- Génération de la documentation html du code java grâce à *javadoc*.
- Répartition du travail à faire pour les 2 semaines de vacances scolaires.
- Programmation: réalisation du *Meeting Agent*, l'agent de prise de RDV.

Période Janvier-Février 1999

- Programmation: Réalisation des différentes méthodes des agents.
- Programmation: Connexion de la partie serveur avec la partie client qui comporte principalement l'interface.
- Programmation: Tests et corrections.

Réunion Février 1999

- Dernières mises-au-point avec M. Benhamou et préparation de la soutenance.

Soutenance de TPED Février 1999

Conclusion

Cela a été difficile.

L'enveloppe horaire d'un peu plus de 2 mois était largement insuffisante pour ce que nous avons voulu faire.
La conception nous a réclamé plusieurs semaines, mais ce n'est rien comparé au temps nécessaire à la programmation.
En fait, nous avons travaillé sur ce projet dès que nous le pouvions, pendant les pauses déjeuners, chez nous le soir, ...

Nous avons été gênés par le manque d'ordinateurs (2 machines à mis-temps au lieu de 3 initialement prévues), qui additionné à la topologie sécurisée du réseau, et les problèmes de *sockets* de la machine *nef.esiea.fr* nous permettaient de ne travailler qu'à un seul programmeur à la fois.

Puis il y a eu les bugs de *Voyager* que nous ne pouvions prévoir. Ce logiciel que nous avons décidé d'utiliser pour nous décharger d'une partie de la programmation n'est pas encore parfait. Les serveurs ne peuvent être adressés que par leur adresse IP. *Voyager* ne fait pas de résolution de noms. Trouver ce bug nous a coûté beaucoup, aussi bien au niveau du temps qu'au niveau du moral.
Donc à posteriori nous n'aurions peut-être pas du utiliser ce produit tout fait, mais plutôt tout programmer nous même en utilisant Java Remote Method Invocation (RMI) et CORBA.
Cette constatation va à l'encontre de la politique d'utilisation de l'existant. Ainsi l'existant doit-il être utilisé lorsqu'il a fait ses preuves. L'expérience *Voyager* n'en a été que plus enrichissante.

Nous pouvons conclure que la conception et la réalisation de ce projet nous ont été très formatrices. Nous nous sommes en effet attachés à obtenir un résultat le plus professionnel possible, et ce aussi bien au niveau de l'architecture, des performances, que de l'évolutivité du produit et de la documentation des programmes.

Nous tenons à remercier Philippe Benhamou pour le temps et l'attention qu'il nous a consacré.

Annexes techniques

Le MiddleWare

6.7 Présentation

Les utilisateurs de systèmes d'information ont de plus en plus besoin d'échanger et de partager des informations. Ces informations peuvent être de nature très différentes et sous des formats qui le sont tout autant (fichiers textes, images, messages, données, etc...). Et, plus récemment ces informations peuvent être situées à des endroits différents, c'est à dire sur des postes distants, sur des réseaux différents ou encore sur l'Internet.

La diversité et la spécialisation des supports utilisés sont inversement proportionnelles à la facilité que les utilisateurs ont à se les échanger. De plus, il existe des milliers d'applications indépendantes écrites dans des quinzaines de langages différents sur des dizaines de systèmes d'exploitation différents.

C'est de ce manque de standards et du besoin vital de communiquer (surtout pour les entreprises), qu'est né un nouveau type de composants logiciels : les *MiddleWare*.

Concrètement, un Middleware est un ensemble de programmes qui isole une application de l'ensemble des processus qui résultent de son lancement sur le système.

Un MiddleWare permet la communication entre des clients et des serveurs ayant des structures et une implémentation différentes. Il permet l'échange d'informations dans tous les cas et pour toutes les architectures. Enfin, les MiddleWare doivent fournir un moyen aux clients de trouver leurs serveurs, aux serveurs de trouver leurs clients et en général de trouver n'importe quel objet atteignable.

La plupart des MiddleWare sont conformes aux spécifications *CORBA*.

6.8 CORBA

CORBA est la spécification d'une architecture orientée objet pour applications. Elle a été définie par l'OMG dans un document publié au mois de novembre 1990.
Avec CORBA, le client et le serveur sont formellement séparés, on peut alors changer l'un sans changer l'autre. Ainsi le client qui utilise une interface CORBA n'a pas besoin de connaître la façon dont le serveur va exécuter la tâche mais seulement comment faire une requête pour qu'une opération soit effectuée sur un objet.

Le Broker

Dans un environnement classique de client/serveur, il existe une relation de tête-à-tête entre les clients et leurs serveurs. CORBA ajoute un intermédiaire entre le client et le serveur: l'Object

Request Broker, ou *Broker*. Le broker possède " l'intelligence " nécessaire pour mettre en rapport la requête provenant d'un client avec les serveurs qui peuvent y répondre.

L'apport d'un broker entraîne plusieurs améliorations :

- Le client et le serveur n'ont plus besoin de se connaître de façon directe. Ils se trouvent grâce au broker. Ainsi seul le broker a besoin de connaître la localisation et les ressources disponibles du client ou du serveur qui se trouvent sur le réseau.
- Une relation en tête-à-tête entre les clients et leurs serveurs n'est plus requise. Ainsi grâce au broker, plusieurs serveurs peuvent travailler avec un seul client, ou un seul serveur peut travailler avec plusieurs clients.
- Une application cliente peut localiser et inter-réagir avec un objet ou un serveur durant une transaction. Dans l'environnement classique client/serveur, la requête est prédéfinie alors qu'ici, le client peut invoquer une requête sur un objet durant la transaction.

La requête

CORBA sépare le client du serveur en restreignant la communication qui peut exister entre eux par un type de message appelé requête. Chaque interaction dans un système CORBA est :

- soit un client qui envoie une requête, c'est à dire une invocation.
- soit un serveur qui répond à une requête.

Toutes les requêtes ont la même structure:

- une indication sur l'opération que doit exécuter le serveur à la demande du client,
- une référence spécifique à l'objet sur lequel le serveur doit effectuer l'opération,
- un mécanisme qui doit permettre de retourner un message concernant la réussite ou l'échec de l'opération,
- une référence optionnelle à un objet de contexte,
- et des arguments spécifiques à l'opération à effectuer.

IDL

Le client et le serveur ont besoin d'informations pour pouvoir travailler sur les mêmes objets. Par exemple, chaque objet doit annoncer les opérations que l'on peut effectuer sur lui.
CORBA dispose ces informations dans une interface qui définit les caractéristiques et les comportement de chaque type d'objet, y compris les opérations qu'un serveur peut exécuter sur ces objets. Pour définir une interface, on utilise l'IDL (Interface Definition Language) pour coder l'information dans un jeu de définitions d'interfaces. Les développeurs entreposent ces définitions d'interfaces dans des fichiers IDL.

Ainsi avant d'écrire une application cliente ou serveur, on doit d'abord définir son ou ses interface(s), c'est à dire créer un fichier IDL. C'est ce fichier qui contient les définitions des interfaces que le client ou le serveur supporte.

L'IDL est un langage de définition et non un langage de programmation grâce auquel on définit des interfaces et des structures de données et non des algorithmes. On utilise des fichiers IDL pour générer un code source pour le langage de programmation désiré. On peut donc, par exemple, écrire des applications clientes en LISP et des applications serveurs en C et faire communiquer ce client et ce serveur.

6.9 Voyager

Une plate-forme Multi-Agents

Voyager est un middleware développé par ObjectSpace. Voyager implémente une plateforme pour systèmes distribués et est 100% Java. De plus, il est utilisable gratuitement.

Voyager inclut un ORB supportant les objets mobiles et les Agents Autonomes.
Les services fournis sont entre-autres la persistance, la communication de groupes scalaires et une gestion basique des services Directory.

Un des avantages de Java est de charger en cours de fonctionnement des classes dans sa Machine Virtuelle (VM). Cela permet d'utiliser des objets mobiles et des agents autonomes comme outils pour construire des objets distribués.

C'est pour nous décharger d'une part de programmation que nous avons choisi Voyager.

Utilisation

Voyager nous a donc servit à :

- Les Agents sont déplacés de serveurs en serveurs.
- Des méthodes sont invoquées sur les Agents via leur *proxy*.
- Les Agents sont rassemblés en sous-groupes, de manière à invoquer des méthodes sur des sous-groupes, plutôt que sur chacun des Agents.

Java

Compilation

Nous avons écrit toutes les parties de l'ensemble client/server Orchid sous la forme de *packages*. Ainsi chaque fichier *.java appartient à un package.

La structure des *packages* est la suivante :

```
orchid-+
       |-agent
       |-doc
       |-gui
       |-object
       `-server
```

Ainsi nous avons les *packages* orchid.agent, orchid.gui, orchid.object et orchid.server.
orchid/doc est l'emplacement des fichiers de documentation que nous générons en utilisant *javadoc*.

Pour compiler les fichiers *.java *il faut se placer dans le même répertoire que orchid.*
La commande est alors la suivante :

```
javac orchid/agent/*.java orchid/gui/*.java orchid/object/*.java orchid/server/*.java
```

Exemple :

```
I5:~/javaProgs$ javac orchid/agent/*.java orchid/gui/*.java orchid/object/*.java
orchid/server/*.java
```

Pour exécuter une classe *il faut toujours se placer dans le même répertoire que orchid.*
Exemples :

```
I5:~/javaProgs$ java orchid.object.dateTest
I5:~/javaProgs$ java orchid.gui.connect
I5:~/javaProgs$ java orchid.gui.connect i5.esiea.fr
I5:~/javaProgs$ java orchid.gui.connect nef.esiea.fr
```

Génération de documentation

Il est possible de générer une documentation au format *html* en utilisant la commande *javadoc*. Pour exécuter la commande *javadoc*, *il faut toujours se placer dans le même répertoire que orchid.*

La commande que nous utilisons est la suivante :

```
javadoc -author -d orchid/doc/ orchid.agent orchid.gui orchid.object orchid.server
```

Cela génère les fichiers html dans le sous-répertoire orchid/doc. L'option *-author* affiche le nom du membre de l'équipe qui a écrit telle ou telle classe.
En résumé l'utilisation est la suivante :

```
javadoc <-author> <-d where_to_put_the_doc> package_name1 package_name2 ...
```

Exemple :

```
I5:~/javaProgs$ javadoc -author -d orchid/doc/ orchid.agent orchid.gui orchid.object
orchid.server
```

Attention : Il est obligatoire de générer la documentation pour tous les packages et fichiers en même temps. L'utilisation de *javadoc* écrase en effet tous les fichiers .html qui pourraient être déjà présents.

Voyager

Sur les serveurs, Voyager est lancé sur les ports 10000 et plus.

Sur les clients, Voyager est lancé sur le port 11000.

`servers_list.cfg` contient une liste, mise à jour à chaque connection, des serveurs dans l'ordre desquels ils doivent être contactés par le client.

SQL

Création de la base de données orchid

```
createdb orchid_db
```

Remplissage de la base de données orchid

```
psql orchid_db

orchid_db=> create table servers (name text, url text, availability int, master boolean);

orchid_db=> insert into servers values ('i5','i5.esiea.fr',10,TRUE);
orchid_db=> insert into servers values ('nef','nef.esiea.fr',4,FALSE);

orchid_db=> select * from servers;

orchid_db=> create table users (name text, password text,
                            info text, profile text, online boolean);

orchid_db=> insert into users values ('philippe.benhamou','a23q','ONERA','',TRUE);
orchid_db=> insert into users values ('antoine.souliez','79ty','ESIEA','',FALSE);
orchid_db=> insert into users values ('ma.darche','12','ESIEA','',TRUE);

orchid_db=> select * from users;

orchid_db=> create table agents (name text, owner text, home text,
```

```
                              currentlocation text, frozen boolean);

orchid_db=> insert into agents values ('ma__philippe.benhamou','philippe.benhamou',
                              'i5.esiea.fr','nef.esiea.fr',FALSE);
orchid_db=> insert into agents values ('ma__antoine.souliez','antoine.souliez',
                              'i5.esiea.fr','nef.esiea.fr',FALSE);
orchid_db=> insert into agents values ('ma__ma.darche','ma.darche',
                              'i5.esiea.fr','i5.esiea.fr',FALSE);

orchid_db=> select * from agents;
```

```
orchid_db=> create table meetings (user text, subject text, note text,
                                   initiator text, date date, timeofday int,
                                   duration int, participants text, frozen boolean);

orchid_db=> insert into meetings values ('philippe.benhamou','réunion TPED',
                                         'définition du terme agent',
                                         'ma.darche','1/10/1998',1430,4,
                                         'antoine.souliez frédéric.salvy
                                          emmanuel.pierre mo.varroy ma.darche',
                                          FALSE);

orchid_db=> select * from meetings;
```

Suppression des tables de la base de données

Il faut des fois passer par cette étape lorsqu'il est nécessaire de changer le type de certains tuples.

```
orchid_db=> drop table servers;
orchid_db=> drop table users;
orchid_db=> drop table meetings;
orchid_db=> drop table agents;
```

Suppression de la base de données

Cette opération n'est pas utile dans le cadre du projet, mais ce trouve ici à titre d'information.

```
destroydb orchid_db
```

Requêtes avancées

```
psql orchid_db -c "select * from agents where owner
                                         in ('antoine.souliez','ma.darche')"


psql orchid_db -c "select * from users order by name"

psql orchid_db -c "select * from meetings order by date"
```

Glossaire

agent
Un *agent* est une entité (logicielle, animale, humaine, etc ...) autonome. Cette entité peut prendre des initiatives et agir sur elle-même et/ou son environnement.

agents mobiles
Des *agents mobiles* sont des *agents* capables de se déplacer. Dans notre cas ils se déplacent de serveur en serveur sur le réseau suivant leurs besoins.

applet
Une *applet* est une application Java qui a été compilée pour une utilisation dans un document HTML. Ces applications sont encapsulées par une couche de sécurité afin ne pas être corrompues par d'autres programmes.

bêta (version bêta)
Une application est dite *bêta* (ou en *version bêta*) quand elle est en cours de développement, et donc pas forcément stable. Elle est souvent diffusée gratuitement afin que les utilisateurs puissent la tester et apporter leurs remarques, de manière à en aider l'amélioration. La mise à disposition des utilisateurs de versions bêta est aussi un bon outil commercial dans le sens où il donne envie du produit final.

chat
Le *chat* est une application permettant à plusieurs personnes sur un réseau de dialoguer en ligne dans une salle virtuelle dite *chatroom*. Les personnes commmuniquent au moyen de texte saisi au clavier. Le texte est envoyé par le client au serveur de *chat*.

client
Any code which invokes an operation on a distributed object.
A client might itself be a CORBA object.

client stub
A Java programming language class generated by idltojava and used transparently by the client ORB during object invocation. The remote object reference held by the client points to the client stub. This stub is specific to the IDL interface from which it was generated, and it contains the information needed for the client to invoke a method on the CORBA object that was defined in the IDL interface.

client tier
The portion of a distributed application that requests services from the server tier. Typically, the client tier is characterized by a small local footprint, a graphical user interface, and simplified development and maintenance efforts.

cluster (PCs cluster)
Un *cluster* de PCs est composé de plusieurs unités centrales de PCs reliées ensembles et fonctionnant comme une seule machine. Cela permet d'obtenir une machine plus puissante que ce que la technologie permet de réaliser. Seulement, il faut un système d'exploitation qui puisse gérer ce type de situation. Linux a naturellement été choisi pour ce type d'architecture machine.

Common Object Request Broker Architecture (CORBA)
An OMG-specified architecture that is the basis for the CORBA object model. The CORBA specification includes an interface definition language (IDL), which is a language-independent way of creating contacts between objects for implementation as distributed applications.

CORBA object
An entity which (1) is defined by an OMG IDL interface, and (2) for which an object reference is available. Object is also the implicit common base type for object references of IDL interfaces.

DARPA
Defense Advanced Research Projects Agency.
The Internet has first been created by this US agency before it starts to grow by itself unregulated.

data store tier
The portion of a distributed application that manages access to persistent data and its storage mechanisms, such as relational databases.

distributed application
A program designed to run on more than one computer, typically with functionality separated into tiers such as client, service, and data store.

distributed environment
A network of one or more computers on which objects are installed on the various machines and can communicate with each other.

Dynamic Invocation Interface (DII)
An API that allows a client to make dynamic invocations on remote CORBA objects. It is used when at compile time a client does not have knowledge about an object it wants to invoke. Once an object is discovered, the client program can obtain a definition of it, issue a parameterized call to it, and receive a reply from it, all without having a type-specific client stub for the remote object.

Dynamic Skeleton Interface (DSI)
An API that provides a way to deliver requests from an ORB to an object implementation when the type of the object implementation is not known at compile time. DSI, which is the server side analog to the client side DII, makes it possible for the application programmer to inspect the parameters of an incoming request to determine a target object and method.

factory object
A CORBA object that is used to create new CORBA objects. Factory objects are themselves usually created at server installation time.

FTP
File Transfer Protocol

groupware
Voir *workflow*.

HTTP
Hypertext Transfer Protocol

IDE
Integrated (or Interactive) Development Environment.
Généralement un environnement de développement graphique. Les outils *RAD* sont généralement des IDE.

idltojava compiler
A tool that takes an interface written in OMG IDL and produces Java programming language interfaces and classes that represent the mapping from the IDL interface to the Java programming language. The resulting files are .java files.

IDL attribute
That part of an IDL interface that is similar to a public class field or C++ data member. The idltojava compiler maps an OMG IDL attribute to accessor and modifier methods in the Java programming language. For example, an interface ball might include the attribute color. The idltojava compiler would generate a Java programming language method to get the color, and unless the attribute is readonly, a method to set the color. CORBA attributes correspond closely to JavaBeans properties.

IDL exception
An IDL construct that represents an exceptional condition that could be returned in response to an invocation. There are two categories of exceptions: (1) system exceptions, which inherit from org.omg.CORBA.SystemException (which is a java.lang.RuntimeException), and (2) user-defined exceptions, which inherit from org.omg.CORBA.UserException (which is a java.lang.Exception).

IDL parameter
One or more objects the client passes to an IDL operation when it invokes the operation. Parameters may be declared as "in" (passed from client to server), "out" (passed from server to client), or "inout" (passed from client to server and then back from server to client).

implementation
A concrete class that defines the behavior for all of the operations and attributes of the IDL interface it supports. A servant object is an instance of an implementation. There may be many implementations of a single interface.

initial naming context
The NamingContext object returned by a call to the method orb.resolve_initial_references ("NameService"). It is an object reference to the COS Naming Service registered with the ORB and can be used to create other NamingContext objects. See also: naming context

interface
The java language supports a feature called *interfaces*. An *interface* contains no code. Instead, it defines a set of method signatures that must be defined by any class that implements the *interface*.

Interface Definition Language (IDL)
The OMG-standard language for defining the interfaces for all CORBA objects. An IDL interface declares a set of operations, exceptions, and attributes. Each operation has a signature, which defines its name, parameters, result and exceptions. OMG IDL does not include implementations for operations; rather, as its name indicates, it is simply a language for defining interfaces.

Interface Repository (IFR)
A service that contains all the registered component interfaces, the methods they support, and the parameters they require. The IFR stores, updates, and manages object interface definitions. Programs may use the IFR APIs to access and update this information. An IFR is not necessary for normal client/server interactions.

Internet InterORB Protocol (IIOP)
The OMG-specified network protocol for communicating between ORBs. Java IDL conforms to IIOP version 1.0.

invocation
The process of performing a method call on a CORBA object, which can be done without knowledge of the object's location on the network. Static invocation, which uses a client stub for the invocation and a server skeleton for the service being invoked, is used when the interface of the object is known at compile time. If the interface is not known at compile time, dynamic invocation must be used.

Java IDL
The classes, libraries, and tools that make it possible to use CORBA objects from the Java programming language. The main components of Java IDL are an ORB, a naming service, and the idltojava compiler. The ORB and naming service are part of JDK1.2; the idltojava compiler can be downloaded from the Java Developer Connection (JDC) web site.

login
Le *login* est un mot anglais qui sert à désigner la demande de connection à une application. En général, on répond à un *login* en entrant un nom d'utilisateur puis un mot de passe pour autoriser l'accès à une application.

middleware
Un *middleware* permet la communication entre des clients et des serveurs ayant des structures et une implémentation différentes. Il permet l'échange d'informations dans tous les cas et pour toutes les architectures.

Most-Constrained Participant (MCP)
Le *Most-Constrained Participant* est le participant à une réunion qui a le plus de contraintes. C'est à dire celui qui a l'emploi du temps le plus chargé.
Le calcul du *MCP* doit être fait avant chaque négociation, de manière à l'optimiser. Inutile en effet de négocier sur des plages horaires non disponibles pour certains participants.

name binding
The association of a name with an object reference. Name bindings are stored in a naming context.

namespace
A collection of naming contexts that are grouped together.

naming context
A CORBA object that supports the NamingContext interface and functions as a sort of directory which contains (points to) other naming contexts and/or simple names. Similar to a directory structure, where the last item is a file and preceding items are directories, in a naming context, the last item is an object reference name, and the preceding items are naming contexts.

naming service
A CORBA service that allows CORBA objects to be named by means of binding a name to an object reference. The name binding may be stored in the naming service, and a client may supply the name to obtain the desired object reference.

object
A computational grouping of operations and data into a modular unit. An object is defined by the interface it presents to others, its behavior when operations on its interface are invoked, and its state.
object implementation See implementation.

Object Management Group (OMG)
An international organization with over 700 members that establishes industry guidelines and object management specifications in order to provide a common framework for object-oriented application development. Its members include platform vendors, object-oriented database vendors, software tool developers, corporate developers, and software application vendors. The OMG Common Object Request Broker Architecture specifies the CORBA object model. See www.omg.org for more information.

object reference
A construct containing the information needed to specify an object within an ORB. An object reference is used in method invocations to locate a CORBA object. Object references are the CORBA object equivalent to programming language-specific object pointers. They may be obtained from a factory object or from the Naming Service. An object reference, which is opaque (its internal structure is irrelevant to application developers), identifies the same CORBA object each time it is used. It is possible, however, for multiple object references to refer to the same CORBA object.

Object Request Broker (ORB)
The libraries, processes, and other infrastructure in a distributed environment that enable CORBA objects to communicate with each other. The ORB connects objects requesting services to the objects providing them.
operation (IDL) The construct in an IDL interface that maps to a Java programming language method. For example, an interface ball might support the operation bounce. Operations may take parameters, return a result, or raise exceptions. IDL operations can be oneway, in which case they cannot return results (return values or out arguments) or raise exceptions.

package
En Java, un *package* est une entité logique. Son utilité est de lui faire appartenir des classes qui se connaitront alors entre elles. La répartition des classes en *packages* apporte une très grande lisibilité et modularité au code.

persistence
La *persistence* utilisée par *Voyager* est une méthode voisine de la *sérialisation*. Elle 'freeze' une application Java ou un objet présent en mémoire en le stockant sur la mémoire de masse.

PIDL (Pseudo-IDL)
The interface definition language for describing a CORBA pseudo-object. Each language mapping, including the mapping from IDL to the Java programming language, describes how pseudo objects are mapped to language-specific constructs. PIDL mappings may or may not follow the rules that apply to mapping regular CORBA objects.

port
Generally speaking, a computer has a single physical connection to the network. All data destined for a particular computer arrives through that connection. However, the data may be intended for different applications running on the computer. Through the use of *ports*, the computer knows to which application to forward the data.
Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined.
The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network.
Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.
In connection-based communication such as TCP, a server application binds a socket to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then rendezvous with the server at the server's port.

Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers. The port numbers ranging from 0 - 1023 are restricted; they are reserved for use by well-known services such as HTTP and FTP and other system services. These ports are called well-known ports. Your

applications should not attempt to bind to them.

pragma
A directive to the *idltojava* compiler to perform certain operations while compiling an IDL file. For example, the pragma "javaPackage" directs the idltojava compiler to put the Java programming language interfaces and classes it generates from the IDL interface into the Java programming language package specified.

protocole propriétaire
Propriétaire est opposé à *standard.*
Windows est un exemple d'utilisation de protocoles propriétaires : dans ce système tous les standards sont modifiés de manière à piéger le consommateur et lui enlever toute possibilité d'utiliser d'autres produits que ceux proposés par la société Microsoft.
Les protocoles *propriétaires* naissent souvent à cause de logiques commerciales. Ils gênent à la fois les sociétés et les programmeurs.

protocole standard
Tout protocole utilisé, compris et adopté par une vaste communauté de constructeurs et d'utilisateurs. Généralement un protocole devient *standard* lorsqu'il s'est montré commercialement et techniquement meilleur que les autres. Il fait alors souvent l'objet d'une norme.

proxy
Un proxy est une application qui s'occupe de la redirection de tout ce qui lui est envoyé. De cette manière, on peut toucher plusieurs entités virtuelles sans connaitre leur emplacement.

pseudo-object
An object similar to a CORBA object in that it is described in IDL, but unlike a CORBA object, it cannot be passed around using its object reference, nor can it be narrowed or stringified. Examples of pseudo-objects include the Interface Repository and DII which, although implemented as libraries, are more clearly described in OMG specifications as pseudo-objects with IDL interfaces. The IDL for pseudo-objects is called "PIDL" to indicate that a pseudo-object is being defined. servant object An instance of an object implementation for an IDL interface. The servant object is registered with the ORB so that the ORB knows where to send invocations. It is the servant that performs the services requested when a CORBA object's method is invoked.

PUSH/PULL
Les méthodes de *PUSH* et de *PULL* définissent 2 manières d'accéder à l'information. *PULL* (le plus commun) : le client va chercher les informations sur le serveur. *PUSH* : le serveur envoi l'information au client.

RAD
Rapid Application Development.
It is an approach to building computer systems which combines tools and techniques, user-driven prototyping, and stringent project-delivery time limits into a potent, tested, reliable formula for top-notch quality and productivity.
RAD raises the quality of finished systems while reducing the time it takes to build them.

server
A program that contains the implementations of one or more IDL interfaces. For example, a desktop publishing server might implement a Document object type, a ParagraphTag object type, and other related object types. The server is required to register each implementation (servant object) with the ORB so that the ORB knows about the servant. Servers are sometimes referred to as object servers.

server skeleton
A public abstract class generated by the idltojava compiler that provides the ORB with information it needs in dispatching method invocations to the servant object(s). A server skeleton, like a client stub, is specific to the IDL interface from which it is generated. A server skeleton is the server side analog to a client stub, and these two classes are used by ORBs in static invocation.

service tier
The portion of a distributed application that contains the business logic and performs most of the computation. The service tier is typically located on a shared machine for optimum resource use.

socket
A *socket* is one end-point of a two-way communication link between two programs running on the network.

stringified object reference
An object reference that has been converted to a string so that it may be stored on disk in a text file (or stored in some other manner). Such strings should be treated as opaque because they are ORB-implementation independent. Standard object_to_string and string_to_object methods on org.omg.CORBA.Object make stringified references available to all CORBA Objects.

Swing
Swing est un ensemble de classes Java permettant l'utilisation d'objets graphiques de type *widget*.

TCP
Transport Control Protocol

trading
Le *trading* est un mot anglais désigant le marchandage, la négociation. Ici, le *trading* désigne l'emploi d'un algorithme de négociation entre agents pour l'échange de données. Un agent va faire du *trading* avec un autre agent pour arriver (si possible) à un commun accord (établir une date de rendez-vous dans un calendrier par exemple).

URL (Uniform Ressource Locator)
Any object on an Internet/Intranet Network is referenced with an URL. It's the location on the net of this file/applet/script/etc.
An URL is written like *<protocole>://<host>.<domain>/<path>*.
For example: *http://www.esiea.fr/usr/people/public_html/index.html*

widget
Composant évolué, prêt à l'emploi et réalisant une ou des fonctions plus complexes. Les *widgets* sont très utilisés pour la construction d'applications graphiques. Par exemple une boîte de dialogue, un bouton sont des *widgets* si un développeur peut les utiliser immédiatement sans avoir à les créer en les programmant.

workflow
Le *workflow*, ou rythme de travail, est une préocupation qui tend à prendre de plus en plus d'importante pour les entreprises.
Des outils logiciels ont ainsi été spécialement développés pour faciliter (et augmenter/rentabiliser) le *workflow*. Le plus connu est sans aucun doute Lotus Notes.
Pour augmenter le *workflow*, il faut toucher au *groupware*. C'est à dire à l'organisation d'une stratégie de groupe. Les domaines favoris des outils de *workflow* et de *groupware* sont le traitement de données, informations et connaissances ainsi que l'organisation des structures et personnels.

Bibliographie

[1] Agent programming. http://www.sikt.hk-r.se/isl/courses/ddv305/laboration/kqml.
Exemples de programmes en Java et KQML pour la simulation de taxis. Résolution de la discussion entre les agents de location et les agents utilisateurs.

[2] The agent society. http://www.agent.org.
Groupe de standardisation des agents logiciels. Différentes approches et définition des agents.

[3] Applications de travail collaboratif. http://www.cru.fr/multimedia.
Le but du travail collaboratif est de permettre la communication audio, vidéo et le partage de support d'affichage au sein d'un groupe de travail.

[4] Autonomous agents conference '97. http://www.isi.edu/isd/AA97/info.html.
La première conférence internationale sur les agents autonomes (Février 1997).

[5] Beowulf project at cesdis. http://www.beowulf.org.
Site internet du cluster Linux de la NASA. Machines massivement parallèles utilisées à la NASA. Evolution et recommandations depuis 1994.

[6] Extreme linux (pcs clusters). http://www.extremelinux.org.
Site internet dédiés aux clusters Linux. Reprise de Beowulf, avec extension et application à la communauté scientifique et à la recherche informatique.

[7] Java-linux. http://wwww.blackdown.org/java-linux.html.
Site internet dédié au port de Java pour Linux. Dernières versions et nouveaux produits Java disponibles sous Linux.

[8] Les agents intelligents. http://crrm.univ-mrs.fr/mannina/mb/agent.htm.
L'autre aspect des agents intelligents dédiés à Internet : robots "intelligents" des moteurs de recherche.

[9] Object management group. http://www.omg.org.
Groupe de normalisation des techniques orientées objets et réseaux (ex : CORBA).

[10] Objectspace (voyager). http://www.objectspace.com.
Grand distributeur de solutions réseaux basées sur Java. Développement d'outils spécialisés comme Voyager et ObjectStore.

[11] Objectstore, voyager and java examples. http://www.meangene.com/java.
Page personnelle dédiée à la programmation en Java des classes ObjectStore et Voyager. Grosse application à un système multi-serveurs.

[12] The olivetti & oracle research laboratory. http://www.uk.research.att.com/omniORB.
Laboratoire de recherche d'ATT. Développement d'un ORB qui implemente les dernières spécifications de CORBA.

[13] Persistent java and mobile web agents. http://copeland.smartchoice.com/~laforge.
Package réalisé par un utilisateur averti pour stocker des objets créés en Java dans des fichiers (persistance).

[14] Sun java technology. http://java.sun.com.
Le site officiel de Java. Tous les derniers produits en téléchargement, documentation et didactiels.

[15] Umbc kqml web. http://www.cs.umbc.edu/kqml.
Les spécifications du langage KQML.

[16] *Component Based Computing and the ObjectStore Cache-Forward Architecture.* Object Design, 1997.
Document d'utilisation du cache d'Objects Persistants ObjectStore.

[17] *Submission IDL/Java Language Mapping.* OMG, March 1997.
Spécification des interfaces Java/CORBA.

[18] *The Common Object Request Broker: Architecture and Specification.* OMG, 1998.
Spécifications CORBA de l'OMG.

[19] *ORBacus for C++ and Java.* Object-Oriented Concepts, 1998.
Documentation de ORBacus, un ORB interfacable en Java.

[20] *Voyager Core Technology 2.0 User Guide.* Objectspace, 1998.
Documentation standard fournie avec les sources.

[21] *Voyager Transaction Service Technical Overview.* Objectspace, 1998.
Documentation sur les services transactionnels de Voyager.

[22] Olivier Aubert. *Introduction à Perl.* 1998.
Cours de Perl en français.

[23] J. Christopher Back. *KQML Users Guide.* 1993.
Documentation de KQML.

[24] de Azevedo H. Barthès JP. Identifying autonomous agents for capitalizing knowledge. Technical report, Laboratoire d'Informatique de Paris 6 (LIP6), 1998.

[25] Jennifer Bigus and Joseph. *Constructing Intelligent Agents with Java.* Wiley Computer Publishing, 1998.
Livre très intéressant sur les applications de l'IA et des SMA en Java.

[26] Christophe Bonnet and Jean-François Macary. *Technologies PUSH.* Fi System, Eyrolles, 1997.
Livre sur les méthodes d'envoi actif de données vers les clients.

[27] Rich Burridge. *Java Shared Data Toolkit User Guide.* Sun Microsystem, June 1998.
Documentation des outils Sun pour iles API Java de travail partagé.

[28] Petrie C. The secretaries nightmare problem, in proceedings of the caia workshop on coordianted design and planning, san antonio, texas. Technical report, 1994.

[29] Weld D. Etzioni O. A softbot based interface to the internet, communication of the acm, 37(7) in lnai 1365 p147. Technical report, 1994.

[30] Jacques Ferber. Eco-problem-solving. Technical report, Institut Blaise Pascal, 1990.

[31] Jacques Ferber. *Les systèmes Multi-Agents.* InterEditions, 1997.
Ce livre décrit exhaustivement tous les systèmes multi-agents, leur conception....

[32] Tim Finin, Jay Weber, et al. *Specification of the KQML, Agent Communication Language.* 1991.

[33] Ketchpel S. Genesereth M. Software agents. Technical report, 1994.

[34] Pattie (Editor) Maes. *Designing Autonomous Agents, Theory and practice from Biology to Engineering and back.* MIT Elsewier, 1991.

[35] Bryan Morgan. *Building distributed applications with Java and CORBA.* Dr Dobb's Journal 284, 1998.

[36] Bernard Moulin. A scenario-based design method and an environment for the development of multiagent systems. Technical report, Laval University, 1995.
Ce rapport présente des étapes d'analyse et de design pour le développement d'un système multi-agent composé d'agent logiciel jouant plusieurs rôles dans des scénari prédéterminés. Un exemple de prise de rendez-vous sert à illustrer la présentation.

[37] Cédric Nicolas, Christophe Avare, and Frédéric Najman. *JAVA Client-Serveur.* Fi System, Eyrolles, 1998.
Développement d'applications critiques d'entreprise, solutions client-serveur. Sont abordés: Java Beans, JDBC, Corba/RMI, Marimba Castanet.

[38] Unruh A. Nodin M. Facilitating open communication in agent systems: The infosleuth infrastructure, in intelligent agents iv, agent theories, architectures and languages, atal '97 lnai 1365 p281. Technical report, 1997.

[39] R. Orfali, D. Harkey, and J. Edwards. *The Essential Client/Server Survival Guide, Second Edition.* Wiley Computer Publishing, 1996.

[40] S. Rimivasan. *Advanced Perl Programming.* O'Reilly, 1998.

[41] Philip Rousselle. *Dynamic distributed systems in Java.* Dr Dobb's Journal 284, 1998.

[42] Kraus S. Schwartz R. Bidding mechanism for data allocation in multi-agent environments, intelligent agents iv, agent theoties, architectures and languages, atal '97 lnai 1365 p61. Technical report, 1997.

[43] Senjen R. Scott A., Jenkin K. Design of an agent-based, multi-user scheduling implementation, in multi-agents systems, theory, languages and application, lnai 1544. Technical report, 1997.

[44] Kraus S. Shehory O. Formation of overlapping coalition for precedence ordered task execution among autonomous agents, in intelligent agents iv, agent theories, architectures and languages, atal '97 lnai 1365 p147. Technical report, 1997.

[45] Jha S. Sheory O., Sycara K. Multi-agent coordiantion through coalition formation, intelligent agents iv, agent theoties, architectures and languages, atal '97 lnai 1365 p147. Technical report, 1997.

[46] Mathieu P. Taquet A. La négociation dans les systèmes multi-agents. une application à la secrétaire virtuelle. Technical report, Université des Sciences et Technologies de Lille, 1996.

[47] Andreas Vogel and Keith Duddy. *JAVA Programming with CORBA (Second Edition)*. Wiley, 1998.

[48] Jamison W. Acacia: An agent based collaborative framework for heterogenous multiagent systems, in multi-agents systems, methodologies and application, lnai 1286 pp76,91. Technical report, 1997.

[49] Ellis C. Wainer J. Agents in groupware systems. Technical report, Laboratoire d'Informatique de Paris 6 (LIP6), 1998.

[50] Larry Wall. *Programmation en Perl 2e édition.* O'Reilly, 1998.

[51] Mark Watson. *Intelligent JAVA Apllications for the Internet and Intranets.* Morgan Kaufmann Publishers, 1997.

[52] Wong. *Web Client Programming.* O'Reilly, 1998.

Index